Masterarbeit

Modifikation eines PHEV-Serienbatteriesteuerungsmodells zur entwicklungsbegleitenden Funktionsabsicherung

Helmut Pichlhöfer

0530779

Institut für Technische Informatik
Technische Universität Graz

Ein Projekt in Zusammenarbeit mit
MAGNA Steyr Fuel and Battery Systems

Betreuer: A.o.Univ.-Prof. Dipl.-Ing. Dr. techn. Eugen Brenner

Betreuer MSBS: Dipl.-Ing. Dr. techn. Stefan Doczy

Graz, Mai 2013

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am (Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, (date) (signature)

KURZFASSUNG

Auf der Grundlage der vorangegangen Seminarprojektarbeit „Einbindung eines PHEV-Serienbatteriesteuerungsmodells in eine Funktionsentwicklungsumgebung“ soll in Kooperation mit MAGNA Steyr Battery Systems eine Masterarbeit entstehen. Die weiterführende Verwendung der im Seminarprojekt dafür vorbereiteten modellbasierten Steuerungssoftware (*Battery Management System*, BMS) eines *Plug-in Hybrid Electric Vehicle* (PHEV) richtet sich an Simulationen von Fahrzyklen, die das reale Verhalten der Zustände des Batteriesystems am Prüfstand veranschaulichen sollen. Die zu simulierenden Fahrzyklen werden davor beim Test am Prüfstand aufgezeichnet und stellen damit die Quell- und Vergleichsdaten für die Simulationen und Analysen am Entwicklungsrechner bereit. Auf diesem Rechner befindet sich die Simulationsumgebung, in der das dafür angepasste und optimierte BMS, in Verbindung mit einem Zellmodell, das reale Batterieverhalten abbilden wird. BMS und Zellmodell liegen dieser Arbeit in Form von MATLAB/Simulink®-Modellen vor, worin auch die Simulationsumgebung für die Simulation und die abschließende Funktionsanalyse implementiert werden sollen.

Besonderes Augenmerk soll dabei auf die Berechnung des Ladezustandes (*State of Charge*, SOC) gelegt werden, da dessen Wert als wichtigste Rückmeldung im Fahrbetrieb gilt, einem komplexen Berechnungsalgorithmus zugrunde liegt und weitere Funktionalitäten der Batterie maßgeblich beeinflusst.

Des Weiteren ist mit dem Steuerungsmodell der Einsatz an Prototypenbatterien unterschiedlicher Zellen vorgesehen. Es benötigt für seine Funktionalität eine Vielzahl von Parametersätzen, welche für eine Reihe von unterschiedlichen Betriebsbedingungen, beispielsweise dem Betrieb über mehrere Temperaturbereiche, zu ermitteln sind. Das geschieht in der Regel am Prüfstand in genau definierten und strukturierten Testabläufen, sogenannten Zellcharakterisierungstests, welche einen hohen zeitlichen Aufwand erfordern. Deswegen ist es notwendig, für spezifizierte, eingeschränkte Betriebsbereiche der Prototypen, reduzierte Parametersätze zu finden, welche die Funktionalität der Steuerung in ausreichender Form sicherstellen und den zeitlichen Aufwand der Zelltests verringern. Da mit weniger Parametern weniger Genauigkeit und Qualität der Modellausgangssignale zu erwarten sind, soll diese Reduktionsauswirkung dargestellt werden. Als Referenz wird der vorhandene, vollständige Parametersatz des Steuerungsmodells verwendet. Dieser wird in Anlehnung an die eingeschränkte Funktionalität der Prototypen reduziert, um die Auswirkungen des Betriebes einer somit fiktiven Prototypenbatterie analysieren zu können.

ABSTRACT

Based on the previous seminar project “Einbindung eines PHEV-Serienbatteriesteuerungsmodells in eine Funktionsentwicklungsumgebung”, in cooperation with MAGNA Steyr Battery Systems, a master thesis will be developed. The further use of the prepared *Battery Management System* (BMS) from the seminar project, which is the essential part for controlling the Battery-Pack of a *Plug-in Hybrid Electric Vehicle* (PHEV) is aimed at the simulation of driving cycles, which should reflect the real behavior of the states of the battery system on the test bench. In order to be simulated, driving cycles are recorded before the test on the test bench and enable the development computer to process the source and comparison data for the simulation and analysis. On this computer, there is a simulation environment, in which the adapted and optimized BMS should, in conjunction with a cell model, map the real battery behavior. In this work, BMS and cell models are available in the form of MATLAB/Simulink textsuperscript® models, in which the simulation environment for simulation and analysis of the final function is implemented.
Special attention will be paid to the *state of charge* (SOC) because its value is a very important feedback while driving. It is based on a complex calculation algorithm and influences other functions of the battery significantly.
The BMS is provided for the use with special batteries, containing different prototype cells. For its functionality, it requires several sets of parameters which are to be determined for a number of different operating conditions, for example, the operation on several temperature ranges. They are calculated on the test bench in well-defined and structured testing processes, named cell characterization tests, which take much time for completion. Therefore it is necessary for specified, limited operating ranges of the prototypes, to find reduced sets of parameters, which ensure the functionality of the controller in a sufficient form and reduce the time effort on cell tests. With reduced parameters, less accuracy and quality of the model output-signals is expected. These impacts are to be displayed. As a comparative reference, the existing, full parameter set of the control model is used.

DANKSAGUNG

Diese Masterarbeit entstand am Institut für Technische Informatik in Zusammenarbeit mit MAGNA Steyr Battery Systems. An dieser Stelle danke ich dem Team der Zellmodellierung und Funktionsentwicklung für die umfassende Unterstützung während der Einführung in die Theorie der PHEV-Modellsimulation und bei der Umsetzung dieser Masterarbeit.
Mein spezieller Dank gilt Stefan Doczy. Er hat mir zum einen die Arbeit an der Masterarbeit an einem Büroplatz innerhalb des MAGNA Steyr Firmengeländes ermöglicht, zum anderen wurde ich von ihm während der Umsetzung des praktischen und schriftlichen Teils umfassend unterstützt. Ein großes Dankeschön richte ich auch an Klaus Hochgatterer und an Peter Dämon für die Organisation dieses Projekts und die dafür notwendige Bereitstellung eines Notebooks.
Ebenso danke ich Prof. Eugen Brenner, der diese Masterarbeit am Institut für Technische Informatik betreut und mit Rat und Anregungen zur Seite gestanden hat.
Des Weiteren danke ich meiner Familie, meinen Freunden und Studienkollegen dafür, dass sie mich in dieser sehr interessanten aber zeitweise auch sehr anstrengenden Zeit mit Rat und Verständnis unterstützt haben.

Helmut Pichlhöfer

Graz, im Mai 2013

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BMS	Battery Management System
BMU	Battery Management Unit
BOL	Begin of Life
CAN	Controller Area Network
CC	Constant Current
CD	Charge Depleting Mode
CSC	Cell Supervision Circuit
CS	Charge Sustaining Mode
CV	Constant Voltage
EEPROM	Electrically Erasable Programmable Read-only Memory
EKF	Extended Kalman Filter
EOL	End of Life
ESS	Energy Storage System
GUI	Graphical User Interface
HEV	Hybrid-Electric-Vehicle
HIL	Hardware in the Loop
HVIL	Hazardous/High Voltage Interlock Loop
KiBaM	Kinetic Battery Model
Li-Ion	Lithium-Ionen
LIM	Linear Model
LUT	Look-up Table
MIL	Model in the Loop
NiCd	Nickel-Cadmium
NiMh	Nickel-Metall-Hydrid

NVM	Non Volatile Memory
OCV	Open Circuit Voltage
OEM	Original-Equipment-Manufacturer
PHEV	Plug-in Hybrid Electric Vehicle
SEI	Solid Electrolyte Interphase
SOC	State of Charge
SOH	State of Health
SOP	State of Power
VCM	Voltage Cell Model
VCU	Vehicle Control Unit
VKM	Verbrennungskraftmaschine
VRLA	Valve Regulated Lead Acid
ZM	Zellmodell

1

Einleitung

1.1 Einleitung

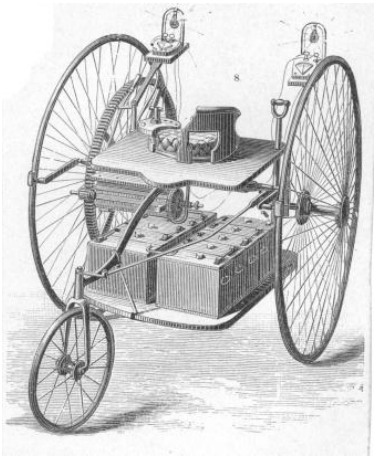

Abbildung 1.1: Abbild des Nachbaus eines 130 Jahre alten Elektro-Dreirads [Emo11]

Die frühesten Aufzeichnungen eines per Elektroantrieb betriebenen Fahrzeugs stammen aus dem Jahr 1828. Der ungarische Erfinder Ányos Jedlik dokumentierte schon Jahre vor schottischen und holländischen Innovatoren die Möglichkeit, ein Gefährt per Elektromotor anzutreiben. Die für automotive Anwendungen nötige Energiespeicherung in einem entsprechenden galvanischen Element hatte in den Jahren um 1842 seine Geburtsstunde, dank der Pionierarbeit des Schotten Robert Davidson und des Amerikaners Thomas Davenport [Bel12].

Zwar waren die ersten Versuche in dieser Technologie vielversprechend, doch

haben sich die ersten Elektroautos mit ihren damals schwerfälligen und unergiebigen Blei-Akkumulatoren [JW06] bald von Fahrzeugen, angetrieben von fossilen Kraftstoffen, verdrängen lassen. Eine nachvollziehbare Technologieentwicklung, wenn man betrachtet, dass Benzin und Diesel damals als günstig, energiereich und reichlich vorhanden galten und Emissionswerte nur selten beachtet wurden. Heute betrachten wir die Situation unserer zukünftigen Mobilität mit ökonomischen und ökologischen Augen, da z.B. der immer weiter steigende Preis des begrenzt vorhandenen Rohöls oder strenger werdende Emissionsgesetze stärker in den Vordergrund treten. So werden wir auch in zahlreichen Informationsquellen daran erinnert, dass der Schritt zu alternativen Antriebsformen unserer Kraftfahrzeuge notwendig ist und schon vor Jahren begonnen hat [Kle09].

Die alternativen Antriebsformen zur Lösung dieser Probleme bilden eine Vielzahl von modernen Forschungs- und Entwicklungsgebieten [Kle09]. Auf eine davon wird in dieser Arbeit näher eingegangen. Konkret handelt es sich um das *Plug-in Hybrid Electric Vehicle* (PHEV), eine Form des Elektroantriebes, ähnlich der des *Hybrid-Electric-Vehicle* (HEV), aber mit der Möglichkeit, die Antriebsbatterie über das örtliche Stromnetz extern zu laden (Plug-in). In diesem Fall wird die Antriebsbatterie gegenüber dem HEV mehr Energieinhalt aufweisen, um damit die Fahrreichweite im Elektromodus zu erhöhen. Der Elektromotor selbst gilt als in vielen Bereich der *Verbrennungskraftmaschine* (VKM) überlegen, vor allem in der Effizienz und dem im Vergleich extrem hohen Antriebsmoment im niedrigen Drehzahlbereich. Dadurch konnte sich diese Antriebsform in einer Reihe von Anwendungsgebieten und Nischenbereichen durchzusetzen, verdrängte die VKM aber nie aufgrund der konkurrenzlosen Energiedichte in den fossilen Energieträgern im Personen- und Lastkraftverkehr [Kle09]. In [Zha10] wird die Ökonomie von PHEVs im Vergleich zu VKM-betriebenen Fahrzeugen diskutiert. Daraus geht hervor, dass PHEVs sehr wohl Vorteile in ihrer Energieeffizienz bieten, sie jedoch nur bei gegebener Leistungsfähig-

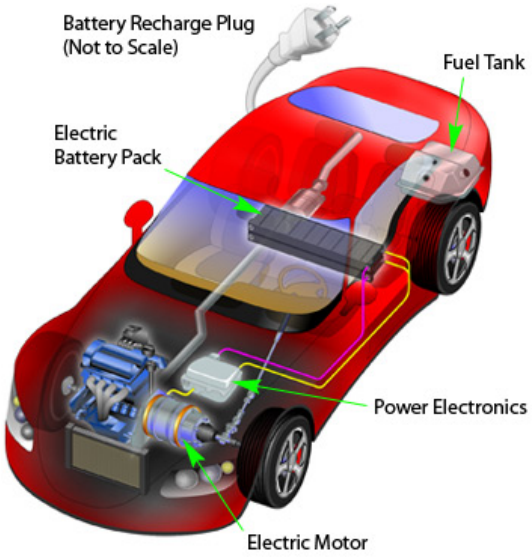

Abbildung 1.2: Schematischer Aufbau eines PHEVs [Cli09]

keit und Infrastruktur des Stromversorger-Netzes ausspielen können, beispielsweise dem bevorzugten Batterieladezeitpunkt bei reduziertem Energiebedarf in den Nachtstunden.
Für den Antrieb des Elektromotors ist es notwendig, die benötigte Energie möglichst dicht und kompakt im Fahrzeug bereitzustellen, ohne die Anforderungen und den Komfort für den alltäglichen Fahrzeuggebrauch zu verringern. Die Speicherung dieser Energie in Batterien stellt aber auch den großen Nachteil der Elektrofahrzeuge dar, da es technologisch noch nicht möglich ist, die Fahrreichweite von VKM-betriebenen Fahrzeugen auch nur annähernd zu erreichen. Der Energieinhalt der einzelnen Zellen und die damit verbundene Fahrreichweite, stellen primäre Ziele der Weiterentwicklung der Antriebsbatterie dar. Diese Entwicklung richtet sich an die in Abschnitt 1.3.1 beschriebenen Antriebsformen und deren unterschiedliche Anwendungsprofile sowie die unterschiedlichen Bau- und Leistungsgrößen, beginnend vom Kleinstwagen, über Oberklasselimousinen bis hin zum Lastkraftwagen.
Im elektrochemischen Sektor der Batterieentwicklung hat sich in den vergangenen Jahren die *Nickel-Metall-Hydrid* (NiMh) Batterie als Hochleistungsbatterie im automotiven Antriebsbereich durchsetzen können [Kle09, JW06]. In den letzten Jahren wurde, nach dem technologischen Durchbruch in der Unterhaltungselektronik, die *Lithium-Ionen* (Li-Ion) Technologie auch für Hochleistungsanwendungen derart weiterentwickelt, dass sie als Ablöse der früher als etabliert geltenden NiMh Batterie gesehen wird. Vor allem in den Bereichen Energie- und Leistungsdichte, welche im Grunde Hauptanforderungen im automotiven Bereich darstellen, konnte sich diese Technologie mittlerweile von den Herkömmlichen absetzen. Dafür wird zur Sicherung der Qualität solcher Serienprodukte noch ein hoher Aufwand nötig sein, um auch noch in den jetzigen Problemfeldern wie Lebensdauer, Sicherheit oder Kostenreduzierung auf akzeptable Ergebnisse zu kommen [Kle09].
Batteriesysteme werden im automotiven Anwendungsbereich per Steuerelektronik (*Battery Management Unit*, BMU) betrieben. Auf der BMU läuft eine umfangreiche Software, um den meist größeren Verband von galvanischen Zellen regeln, überwachen und Informationen über diverse Zustände erhalten zu können. Deren sorgfältige Implementierung stellt für den reibungslosen Betrieb des Systems einen wichtigen Schritt in der Batterieentwicklung dar. Für den Softwareeinsatz am Entwicklungsrechner, am Prüfstand und letzten Endes im Fahrzeug, muss dafür auf geeignete Softwareentwicklungsmethoden zurückgegriffen werden. Die in dieser Arbeit beschriebenen Entwicklungsarbeiten im Anwendungsbereich des PHEVs basieren auf modellbasierter Softwareentwicklung (*Model in the Loop*, MIL) (siehe Abschnitt 1.7), um am Prüfstand (z.B. *Hardware in the Loop*, HIL) diverse Hardware für die Systemfunktionen simulieren zu können. Dabei werden alle relevanten Funktionalitäten

der BMU mit den, für den Testlauf nötigen, Daten versehen und die Funktionen im Modell oder den Hardwarekomponenten beobachtet. Insbesondere beim PHEV muss eine Fülle von komplexen Anforderungen abgedeckt werden können, da in diesem Fall sein Batteriekonzept die schnellen, energiereichen Lade- und Entladezyklen z.B. beim Rekuperieren (Energierückgewinnung), aber auch langzeitliche, niederenergetische Lade- und Entladevorgänge, z.B. die Wiederaufladung an der Ladestation, beherrschen muss [Kle09].
Als weitere Anforderung gilt die Aufteilung der PHEV typischen Betriebsmodi in den rein elektrischen und den kombinierten Modus, wo die VKM als Fahrzeugantrieb dient und vom Elektromotor unterstützt wird, bzw. diesen als Generator benutzt, um damit die Batterie aufzuladen (z.B. beim Bremsen) [Kle09].
Die Software stellt ein physikalisches und chemisches Abbild des realen Zellverbandes dar. Das Verhalten der einzelnen Zellen wird in entsprechenden Test ermittelt und in Form von Systemparametern in das Simulationsmodell eingebunden. Für den Einsatz auf der BMU wird daraus letzten Endes die Software in einer hardwarenahen Programmiersprache generiert. Diese Vorgangsweise gilt für Zellen aus der Serienproduktion genauso wie für jene, die sich in Entwicklung befinden, den Zellprototypen.
Aus der daraus entstehenden Problemstellung, dass die Zellprototypen nicht den kompletten Funktionsumfang benötigen und deswegen auch eine darauf angepasste Software für den Testbetrieb benötigen (siehe Abschnitt 1.7), bildet sich im Rahmen der Zellmodellierung/-Funktionsentwicklung bei MAGNA Steyr Fuel and Battery Systems (siehe Abschnitt 1.2) ein Thema dieser Arbeit.
Die folgenden Kapitel beschreiben den Weg, das Modell einer voll parametrierten Batteriesteuerung für die Steuerung von Zellprototypen zu flexibilisieren und die Funktionalität zu simulieren. Abschnitt 1.3 bietet einen Einblick in die Theorie der Zellmodellierung und Funktionsentwicklung und beschreibt die in dieser Arbeit relevanten Effekte, Begriffe und Funktionen auf Basis des verwendeten PHEV-Batteriesystems. Im Anschluss wird das Modell der PHEV-Batteriesteuerung (siehe Abschnitt 1.5) näher erklärt, zuvor aber ein Einblick in das chemische Verhalten der Li-Ion Zellen (siehe Abschnitt 1.4) gegeben. Abschließend bietet das Kapitel einen Überblick über den aktuellen Stand der Technik (siehe Abschnitt 1.6), ehe es mit der Aufgabenstellung des vorliegenden Projektes endet (siehe Abschnitt 1.7).
Den Entwurf und die Implementierung der Problemstellung beschreibt das darauf folgende Kapitel 2. Es wird mit einer Funktionsanalyse der umgesetzten Funktionalitäten abgeschlossen (siehe Abschnitt 2.4).
Kapitel 3 schließt die Arbeit ab, indem es die beschriebene Thematik zusammengefasst und einen Ausblick über zukünftige Arbeiten in dem Fachbereich bietet.

1.2 MAGNA Steyr Fuel and Battery Systems

Die MAGNA Steyr Fahrzeugtechnik beschäftigt sich schon seit längerer Zeit mit der Aufgabe der Serienbatterie- und Steuerungssoftwareentwicklung im Auftrag unterschiedlicher *Original-Equipment-Manufacturer* (OEM) als Zulieferer und Entwickler. Unter mehreren Standorten in Nordamerika und Europa, hat sich in den Fachbereichen Produktion und Entwicklung von Batterie- und Zellen jener in Graz hervorgehoben, wofür im Zeitraum der Jahre 2005 - 2012 dafür eine eigene Entwicklungs- und Produktionssparte existierte. Heute ist diese Fachbereichssparte wieder in den MAGNA Konzern eingegliedert und bildet gemeinsam mit der Abteilung der Treibstoffkomponenten die MAGNA Steyr Fuel and Battery Systems. Deren Produkte findet man im Lastkraft- und Lieferfahrzeugen und seit kurzem auch im PKW. Dieses Projekt entstand in Zusammenarbeit mit der Abteilung modellbasierte Softwareentwicklung und funktionelle Sicherheit, darin von den Gruppen Zellmodellierung und modellbasierte Softwareentwicklung unterstützt.

1.3 Theoretischer Hintergrund im Fachbereich PHEV

Das folgende Kapitel bietet einen theoretischen Einblick in Funktion und Entwicklung von Batteriesystemen für den Einsatz in Hybridfahrzeugen unterschiedlicher Bauart und Leistungsklassen. Beginnend bei den verschiedenen Varianten der Hybridisierung, werden in weiterer Folge die Softwareentwicklung und ausgewählte Funktionalitäten solcher Systeme erläutert. Hauptaugenmerk wird in diesem Rahmen auf die modellbasierte Entwicklung gelegt.

1.3.1 Hybridisierungsarten des Antriebstranges

Die Hybridisierung von Fahrzeugen wird jeweils in Anwendung und in Elektrifizierungsgrad des Elektroantriebes weiter unterteilt. So werden bei Hybridfahrzeugen, die Elektro- und herkömmlichen Antrieb per VKM vereinen, zwei Varianten unterschieden [Kle09]. Der Antriebsstrang kann in der ersten Variante in serieller, paralleler oder leistungsverzweigter Form verbaut sein. Damit meint man die Kopplung der Energieumsetzer, Elektromotor und VKM, an den Antriebsstrang. Im seriellen Fall wird das Fahrzeug nur vom Elektromotor mechanisch angetrieben und die VKM stellt eine Zusatzenergiequelle, in Form eines Aggregats, z.B. zur Erhöhung der Reichweite, dar (Unterscheidung siehe Abb. 1.3).

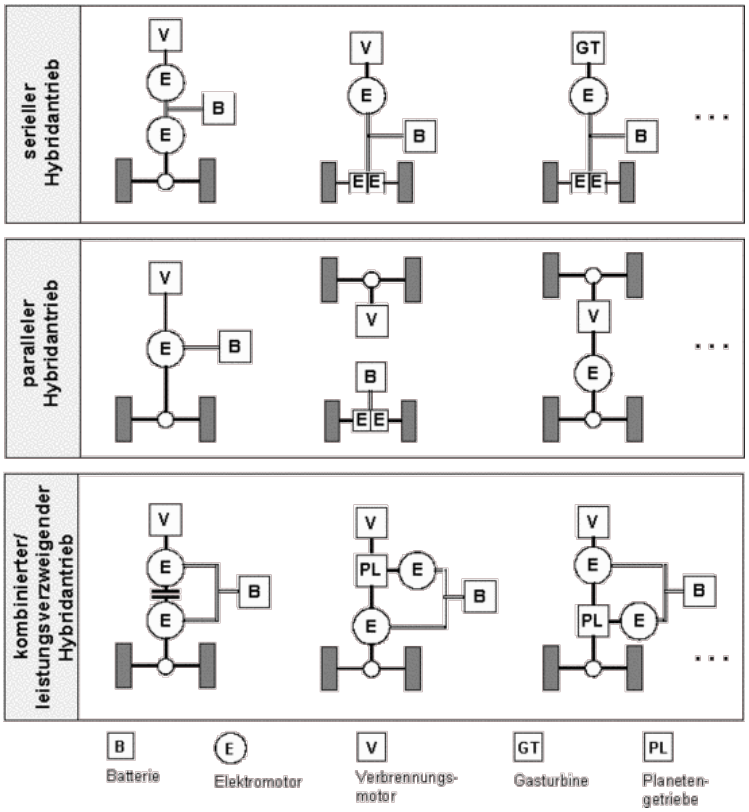

Abbildung 1.3: Die Unterscheidung zwischen seriellem, parallelem und leitstungsverzweigtem Hybridantrieb in schematischer Darstellung [WBBR99]

Für den parallelen und leistungsverzweigten Hybriden gilt, dass beide Motoren für die Kraftübertragung zum Antriebsstrang verwendet werden können. Der leistungsverzweigte Hybrid ist zusätzlich auch in der Lage, kinetische Energie in den elektrischen Energiespeicher zurückzugewinnen. Im parallelen und im leistungsverzweigten Hybriden kommt auch die zweite Variante der Hybridisierung zur Anwendung, der Mikro-, Mild- und Vollhybrid. Dabei wird in der Leistungsanforderung des Elektromotors bestimmt, ob er wie beim Mikrohybrid als Startstopp-Hilfe, beim Mildhybrid als Leistungsunterstützung oder wie beim Vollhybrid als vollwertiger Antriebsersatz für kurze Strecken dienen soll. Abbildung 1.3 zeigt den Zusammenhang der Hybrid-Systeme mit den möglichen Funktionen. Die Elektromotorleistung mit der dafür benötigten Batteriespannung im oberen Teil und den daraus resultierenden Anwendungsbereichen im unteren Teil der Abbildung, stellen eine qualitative Hybrid-Unterteilung nach [Kle09] dar.

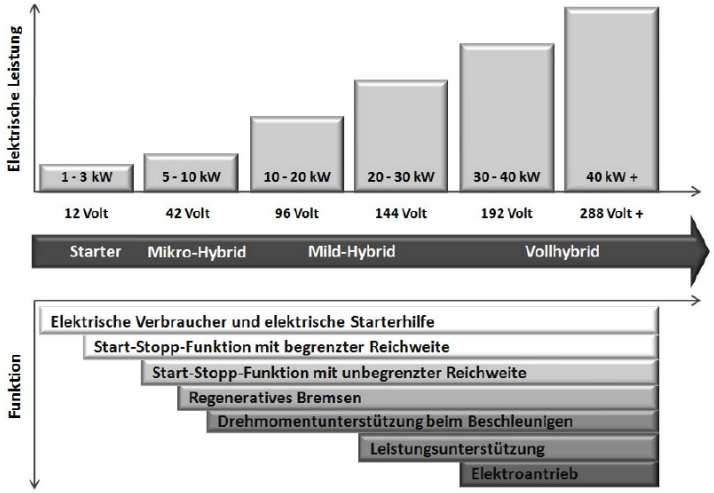

Abbildung 1.4: Die Leistungsklasse des zusätzlichen Elektroantriebes definiert die Hybridisierung des Fahrzeugs [Kle09]

Als zusätzliche Baugruppe gilt die in [GBF10] beschriebene Hybridisierungsart, in der ein HEV zum PHEV erweitert wird, vorgestellt am Beispiel eines Toyota Prius. In diesem Fahrzeug wurde eine zusätzliche Batterie als Plug-In Speicher verbaut, die nur an einer Ladestation aufgeladen werden kann. Die herkömmliche Batterie arbeitet für den HEV-Betrieb, um die Bremsenergie rückgewinnen zu können, die zusätzliche Batterie für den PHEV-Betrieb, um rein elektrisches Fahren zu ermöglichen. Des weiteren wird die VKM auf das erweiterte System angepasst, indem ihr optimaler Betriebsbereich bei niedrigen Motordrehzahlen gesetzt wird, da die restlichen Teillastbereiche mit der elektrischen Energie abgedeckt werden können. [GBF10] will damit zwei, für unterschiedliche Anwendungen konzipierte, Batteriesysteme in einem vereinen.
Die Weiterentwicklung der Batterien für Serienelektroautos und deren Hybridvarianten strebt in erster Linie die Steigerung von zwei maßgebenden Parametern an, die spezifische Energie (Wh/kg) und die Energiedichte (Wh/l). Die spezifische Energie ist ein Maß der Masse des Batteriepakets, ein wichtiges Kriterium, da diese so leicht wie möglich produziert werden sollen. Die Energiedichte bezieht sich letztendlich auf die Möglichkeit, das Batteriepaket kompakt zu produzieren, ein weiteres primäres Ziel in der Autoproduktion. Der Trend geht also in Richtung leichter Fahrzeuge, für hohe Fahrreichweiten und viel nutzbarem Innenraum, um im Alltag als praxistauglich zu gelten [Kle09].

1.3.2 Betriebsarten von Hybridfahrzeugen

In der Literatur findet man sehr häufig die Unterteilung eines Fahrzyklus in den rein elektrischen *Charge Depleting Mode* (CD) und den elektrisch/konventionell kombinierten *Charge Sustaining Mode* (CS) dargestellt (Beispiel in 1.5) [ABK08].

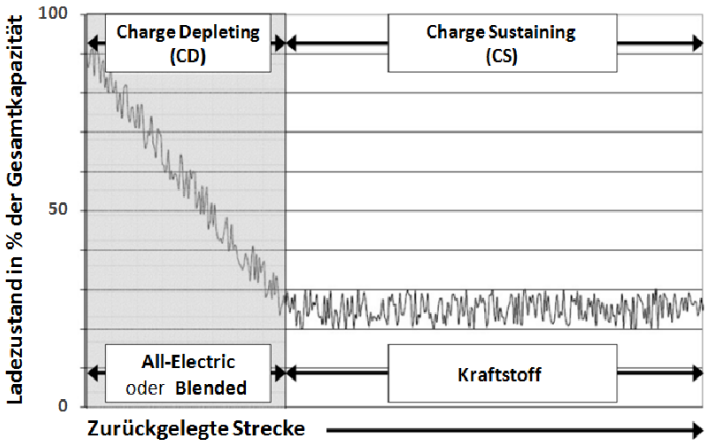

Abbildung 1.5: Der Fahrzyklus wird unterteilt in den vollelektrischen und den hybriden Teil [Kle09]

Um eine treibstoffsparende Betriebsbilanz zu erreichen, muss der Zeitpunkt, wann zwischen den Modi umgeschaltet wird, frei wählbar sein, sodass z.B. auf den für Elektrobetrieb ungünstigen Autobahnfahrten der VKM-Antrieb gewählt wird, um die Batterieladung für den später folgenden Stadtverkehr zu sparen. Die Wahl des Betriebsmodus kann weitgehend unabhängig vom Ladezustand der Batterie erfolgen. Dadurch wird es möglich, dass sich der aktuelle Arbeitspunkt des CS-Modus in hohen oder niedrigen Ladezustandsregionen befindet. In diesen Fällen müssen im Betrieb mögliche Über- oder Tiefentladungen verhindert werden, da die Li-Ion Batterien sehr empfindlich, meist mit dauerhaften Kapazitätsverlust, darauf reagieren [Kle09] (siehe Abschnitt 1.4). Die erwähnte Steuerung und Absicherung der galvanischen Elemente benötigt ein komplexes System aus softwaregesteuerter Elektronik, um kein Sicherheitsrisiko für den Anwender darzustellen. Die Software stellt dafür ein Modell des Verhaltens der realen Batterie dar.

1.3.3 Ansätze der Modellbildung von Batteriesystemen

In der Literatur findet man verschiedene Ansätze, Batteriesysteme für galvanische Elemente (Zelle) wie jene auf der Basis von Blei, Nickel-Metall-Hydrid, Nickel Cadmium oder Lithium-Ionen, zu modellieren [JW06]. Die beschriebenen Verfahren erstrecken sich von experimentellen und elektrochemischen Modellen über analytische Gleichungen und Algorithmen, bis hin zu einfachen bzw. komplexen Ersatzschaltbildern auf Basis elektronischer Bauteile [SMW11, TDD07].
Diese Modellvarianten beziehen sich in der vorliegenden Arbeit ausschließlich auf die Anwendung im Hochleistungssegment, speziell in Fahrzeugen. Sie sind nur teilweise mit Zellen niedriger Leistung vergleichbar, z.B. jene des Unterhaltungsbereichs, wo kleine Zellverbände oder Single-Zellen ihre Anwendung finden. Gemeinsamkeiten haben sie im Temperaturverhalten (siehe Abschnitt 1.4), unterscheiden sich aber aufgrund der Systemspannungen in ihren Sicherheitseinstufungen. Unabhängig von der Leistungsklasse erläutert [HLS11] die Problematik der Handhabung der hohen Anzahl der Einflussfaktoren und des nichtlinearen Spannungsverhaltens der Lithium-Ionen Technologie.
Mit Hilfe von komplexen mathematischen Beschreibungen bilden elektrochemische Modelle das Verhalten von galvanischen Zellen nach, wie laut [She63] aus dem Jahre 1963 hervorgeht. Der hohe Rechenaufwand beruht auf der Anzahl der Einflussfaktoren und den meist nichtlinearen Zusammenhängen solcher Modelle, wie in [SMW11] demonstriert. Ein Beispiel dafür stellt einen Verband von sechs nichtlinearen Differenzialgleichungen dar, der das chemische Verhalten im mikroskopischen Bereich abbildet. Für Echtzeitanwendungen stellt diese Modellvariante, mit teilweise langen Berechnungszeiten, laut [SMW11] keine Alternative dar.
Eine intuitivere Variante stellt das zweite in [SMW11] vorgestellte, stochastische Modell dar, welches auf zeitdiskreten Markov-Ketten basiert. Darin wird die komplette Ladungsmenge der Batterie in Ladungsgruppen eingeteilt, wobei jede einzelne dem Energiebedarf einer speziellen Komponente zugeordnet wird.
Mit den analytischen Modellen wird in [SMW11] eine echtzeitfähige Berechnungsmethode erläutert. Als Grundlage werden die Eigenschaften der Batterie in Form von heuristischen Ansätzen und Formel beschrieben. Beispielsweise wird auf die Peukert'sche Gleichung verwiesen, welche die Abhängigkeit der effektiven Kapazität einer Blei-Batterie vom Entladestrom beschreibt. Ein weiteres analytisches Modell stellt in [SMW11, JH08] das kinetische Batteriemodell in Form von zwei Wassertanks dar, dem sogenannten *Kinetic Battery Model* (KiBaM) [RSKN05, JH08] (siehe Abb. 1.6). Das Fassungsvermögen des kleineren Wassertanks (i) stellt dabei die an der Last verfügbare Kapazität (c) und die Höhe des Füllstandes (h_1) den Ladezu-

stand dar. Der zweite, größere Tank (c-1), beschreibt mit seinem Inhalt (h_2) die mit zeitlichem Versatz verfügbare Energie (j) für den ersten Tank. Dieser Energiefluss zwischen den Tanks wird durch deren Füllstände und die Größe des Ventils (k) beschrieben. Das Auslassventil (R_0) charakterisiert den Innenwiderstand des Modells [RSKN05]).

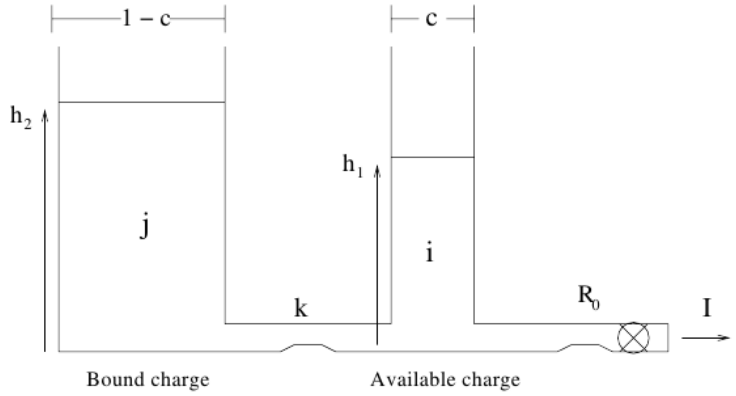

Abbildung 1.6: Skizze des KiBaM [RSKN05]

Die in der Literatur am häufigsten anzutreffende Methode, Batteriesysteme zu modellieren, basiert auf den bereits erwähnten Ersatzschaltbildern auf Basis elektrischer Bauteile. Diese bestehen hauptsächlich aus Konstant-Spannungsquellen, Widerständen und Kondensatoren, weswegen sie auch impedanzbasierte Modelle genannt werden. Die Art und Komplexität des verwendeten Ersatzschaltbildes hängt vom Anwendungsfall der Simulation ab. Es ist fähig, dynamisches Verhalten mit dem Einfluss mehrerer chemischer Eigenschaften, wie die Zelltemperatur unter Belastung oder die Zellalterung, zu berücksichtigen. Temperatur und Alter werden beispielsweise in Form von Parametern bei der Wertedefinition der Bauteile herangezogen. Um Sicherheitsrichtlinien und die Rücksicht auf die Batterie-Lebenszeit in solchen Beschreibungen nicht außer Acht zu lassen, wird in [SMW11] auch auf komplexere Temperaturmodelle verwiesen. Ein besonderer Problemfall darin wäre die bautechnisch bedingte Temperaturschwankung der einzelnen Zellen an verschiedenen Stellen der Batterie.

Eines der Kerngebiete der Batteriesimulation stellt neben der Bestimmung des Zellalters und der Kapazität die Bestimmung des momentanen Ladezustands dar. Eine dafür altbewährte Methode nennt man *Coulomb Counting*. Dieser Begriff stammt von der Vorgehensweise, die Ladung, welche in ein Batteriesystem geladen oder

daraus entnommen wird, über die Zeit mitzuzählen. Ausgehend von einem Start-Ladezustand wird dabei der Strom über die Laufzeit integriert (siehe Abschnitt 1.5.3.5). Dies bildet zwar eine einfache, echtzeitfähige Möglichkeit den Ladezustand zu bestimmen, jedoch nur für begrenzte Zeit. Da jede Messung mit einer gewissen Ungenauigkeit des Sensors erfolgt, ist durch die Summe des Messfehlers pro Abtastung, über eine längere Messdauer betrachtet, mit einer Abweichung des Ladezustandes vom tatsächlichen Wert zu rechnen, weshalb eine gelegentliche Rekalibrierung notwendig wird. Ein weiterer Nachteil dieser Messmethode ist, dass ohne weitere Maßnahme vor Messbeginn keine Aussage über den Initialwert von Ladezustand oder Kapazität gemacht werden kann [CLZH07].

Werden die einzelnen, hier beschriebenen Funktionalitäten zu einer Steuerungssoftware vereinigt, kann damit ein komplexes Batteriesystem betrieben werden.

1.3.4 Das Battery Management System

Auf der Basis der vorgestellten Varianten zur Modellierung von Batteriesystemen ist eine Software, das *Battery Management System* (BMS), im Entwicklungsprozess der MAGNA Steyr Fuel and Battery Systems (siehe Abschnitt 1.2) entstanden. Die Software steuert die Lithium-Ionen-Antriebsbatterie eines PHEVs. Warum diese Zelltechnologie bevorzugt wurde, wird in Abschnitt 1.3.7 erklärt. Die technische Beschreibung des BMS folgt in Abschnitt 2.1.

1.3.5 Das Batteriesystem am Prüfstand

Die in der vorliegenden Arbeit herangezogenen realen Batteriesysteme werden auf Prüfständen der MAGNA Steyr Fuel and Battery Systems (siehe Abschnitt 1.2) getestet [Doc10]. Unter definierten Umständen, den sogenannten Fahr- und Testzyklen, findet die Funktionsüberprüfung von einzelnen Zellen, diversen Zellmodulen oder kompletten Batteriesystemen unterschiedlicher Entwicklungsstufen statt. Getestet werden kann die mechanische Belastungsfähigkeit bis hin zum Missbrauch, das thermische und elektrische Verhalten und die Alterung solcher Komponenten. Im Weiteren werden die letzten drei Punkte näher betrachtet. Dafür besteht der Prüfstand im Wesentlichen aus einer Energiequelle, einer Sicherheits-/Belüftungskammer und einem Kühlmittelaggregat. Die Energiequelle kann über $700V$ DC und $\pm 1000A$

DC bzw. $\pm 250kW$ liefern [Doc10]. Bei den Sicherheits-/Belüftungskammern kommen je nach Anforderung zwei unterschiedliche Baureihen zum Einsatz. Die Klimakammer kann Innenraumtemperaturen zwischen -50° und 100° C erreichen, die Standard-Belüftungskammer, mit Hilfe des Kühlaggregats, bis 0° C kühlen [Doc10]. Das Aggregat, welches max. $40l/min$ Kühlmittel transportieren kann, ist in den Tests für eine rasche Abkühlung bzw. Aufwärmung des Systems auf eine geforderte Umgebungstemperatur verantwortlich. Nachdem es das System auch künstlich erwärmen kann, ist eine simulierte Betriebstemperatur im Bereich von -30° bis 120° C möglich, wie in Abschnitt 2.4 an B-Muster Batteriesystemen (B-Muster siehe Abschnitt 1.3.6) gezeigt wird [Doc10].

1.3.6 Die Bezeichnung der Prototypenmuster

Die Musterstände der System- und Softwareentwicklung werden im Rahmen der MAGNA Steyr Fuel and Battery Systems nach folgender Definition in fünf Stufen gegliedert [MSB12]:

- **Typ A:** solche Softwaremuster besitzen eingeschränkte Funktionalität bzw. Einsatzbereiche, welche in Prüfberichten dokumentiert werden. Hardwarekomponenten, insbesondere Zellprototypen dieses Typs werden in großteils aufwändiger Handarbeit gefertigt und ggf. mit Provisorien für den Betrieb bereitgestellt, weswegen sie als relativ kostspielig gelten. Tests finden nur in kontrollierten Umgebungen mit qualifizierten Personen statt. Komponenten dieses Typs können auch in Versuchsfahrzeugen verbaut werden.

- **Typ B:** dieses Funktionsmuster beschreibt den endgültigen Hardwareentwicklungsstand, aber noch vor der Freigabe der Produktion von Serienwerkzeugen. Die Musterstufe wird nach umfangreichen, erfolgreichen Tests in den Bereichen Lebensdauer und Zuverlässigkeit und der Design-Validierung zugewiesen. Softwaremäßig muss die Funktionalität für Tests am Prüfstand und dem Betrieb im Fahrzeug gewährleistet werden können. Auch gilt eine primäre Erfüllung der Sicherheitsfunktionen als notwendig, wenn auch mit vereinbarten Einschränkungen, die dokumentiert werden müssen.

- **Typ C:** Hardwarekomponenten vom Funktionsmuster Typ C kommen dem Seriendesign schon sehr nahe, weshalb Produktionseinrichtungen auch daran angepasst werden. Von der Funktion her sollten sie die geforderten Punkte des Kunden (in Form eines Lastenhefts) erfüllen. Im Unterschied zum Typ B werden in dieser Stufe alle B-Muster Spezifikationen bzw. Sicherheitsfunktionen

erreicht. Tests finden bereits mit Serienkomponenten am Prüfstand oder mit wenigen, vereinbarten Einschränkungen, im Fahrzeug statt.

- **Typ D:** Die Hardware vom Typ D weist Funktion, Zuverlässigkeit und Lebensdauer laut Lastenheft nach und durchläuft in Vorserienfahrzeugen die letzten Produkt-Validierungen. Für die Software vom Typ D muss die Dokumentation des Produkts fertiggestellt sein.

Der im Allgemeinen als Serienprodukt bekannte **Typ P** bestätigt die Langzeit-Prozessfähigkeit und bedeutet für Software und Hardwarekomponenten die Serienproduktion und den Verkauf an Endkunden.

1.3.7 Vergleich der Zellchemien

Im verwendete PHEV-Batteriesystem dieser Arbeit befinden sich Lithium-Ionen Zellen als Energiespeicher. Warum für zukünftige, automotive Hochstromanwendungen diese Zelltechnologien gegenüber Alternativen, wie jene auf NiMh- oder NiCd-Basis, bevorzugt werden, erklärt [JW06]. Daraus wurde auch die folgende Gegenüberstellung entnommen:

NiMh-Zelle	**Lithium-Ionen-Zelle**
+ Sehr gutes Preis-Leistungs-Verhältnis	+ Höchste Energiedichte
+ Geringe Notwendigkeit für Sicherheitselemente	+ System der Zukunft mit größtem Entwicklungspotential
+ Mittlere Energiedichte und bis auf Hochstromfähigkeit den NiCd-Zellen fast ebenbürtig in Zyklenzahl und Robustheit	+ Notwendigkeit von Elektronik zum Schutz des Batteriepacks, aktuelle Entwicklungen erlauben eine einfachere und kostengünstige Schutzschaltung
− Geringere Lebensdauer bei sehr hohen Strombelastungen oder starker Wärmebelastung	+ Erst am Beginn der Optimierung des Leistungsverhaltens
− Schlechteres Verhalten bei tieferen Temperaturen (> -10° C) im Vergleich zu NiCd-Zellen	*o* Zellpreise sinken stetig, notwendige Sicherheitselektronik verursacht aber zusätzliche Kosten
	− Bisher noch geringere kalendarische Lebensdauer als NiCd- und NiMh-Zellen
	− Sicherheitstests für Batteriepacks nötig zur Erfüllung von Transportvorschriften

In weiter Folge wird das Hauptaugenmerk auf die Funktionsweise und die Steuerung auf Lithium-Ionen Zellen in Batteriesystemen gelegt.

1.4 Einführung in die Lithium-Ionen Technologie

Lithium-Ionen Akkumulatoren bezeichnen wiederaufladbare, chemische Energieträger, welche auch unter dem Begriff „sekundäre Batterien“ in der Literatur zu finden sind. Im Gegensatz dazu können primäre Batterien einmalig bei der Herstellung aufgeladen werden. Das Funktionsprinzip basiert auf Ein- und Auslagerung von Lithium-Ionen in die Kristallgitterstruktur von zwei getrennten Aktivmaterialien, der Anode und der Kathode, die in der Literatur auch als positive und negative Elektroden bezeichnet werden [JW06]. Innerhalb eines Gehäuses bilden sie mit dem Elektrolyt und dem Separator die galvanische Zelle [Bot12]. Die Aktivmaterialien können beim Entladevorgang Elektronen und Lithium-Ionen abgeben, bzw. beim Laden in die Struktur aufnehmen. Die Fähigkeit, Elektronen pro Volumen- oder Flächeneinheit aufzunehmen bzw. abzugeben, wird mit spezifischer Ladung und der Ladungsdichte beschrieben, wie in der Gegenüberstellung der Technologien aus (siehe Abschnitt 1.3.7) gezeigt. Um zwischen den Elektroden eine elektrische Spannung aufbauen zu können, werden dafür Materialien mit einer anwendungsspezifischen Elektroden-Potentialdifferenz gewählt, bspw. wird in Abb. 1.7 die chemische Verbindung Lithium-Metall (0 V) als Referenzpotenzial herangezogen. Je nach Anwendung kann mit der Wahl der Materialien die entstehende Elektrodenspannung beeinflusst werden.

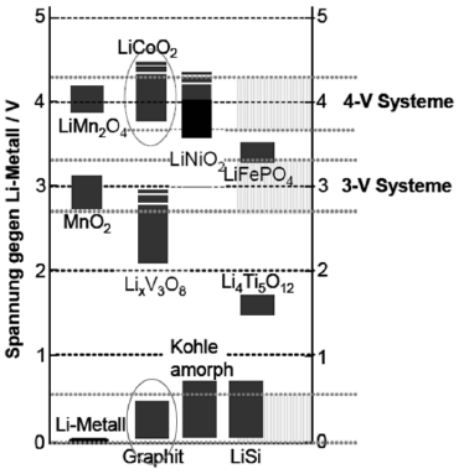

Abbildung 1.7: Gegenüberstellung der Li-Ion Materialien in Bezug auf Li-Metall [Kle09]

Zusätzlich muss auf die Aktivmaterialkapazität, die Sicherheit, die Umweltverträglichkeit und die Stabilität der Elektroden im Hochleistungsbereich, speziell in Fahrzeug-Antriebsbatterien, bei der Wahl der Materialien geachtet werden [JW06]. Den Lithium-Transfer zwischen den Elektroden ermöglicht ein Elektrolyt, in dem sich zusätzlich ein Separator, zur räumlichen Trennung der Pole, befindet (siehe Abb. 1.8).

Betrachtet man den Entladevorgang einer Zelle, findet an der Anode (negativ geladene Elektrode) die sogenannte Oxidation statt. Die frei werdenden Elektronen streben nach Schließen des externen Stromkreises danach, das Potential

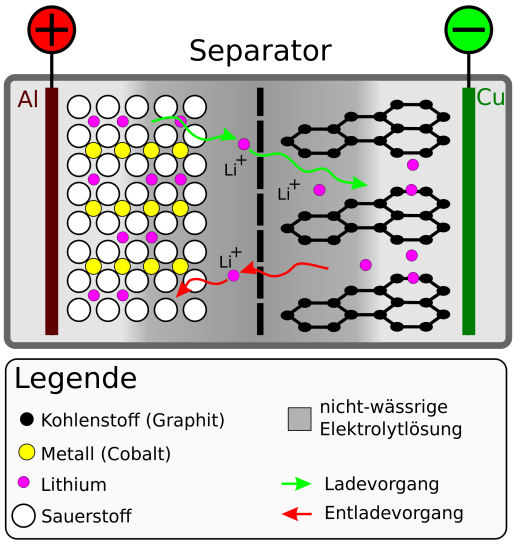

Abbildung 1.8: Aus [Wik12]: „Schematischer Aufbau einer Lithium-Ionen-Zelle (positive Elektrode: LiCoO2; negative Elektrode: Li-Graphit).“

der Elektroden auszugleichen, in dem sie über eine externe Verbindung zur Kathode fließen und dabei elektrische Arbeit verrichten können.
Dieses Streben nach Ladungsneutralität veranlasst die in der Anode eingelagerten Lithium-Ionen dazu, durch den Elektrolyt zur Kathode zu wandern. Elektronen und Lithium-Ionen lagern sich in der Wirtsstruktur der Kathode ein, welche eine hohe Ionisierung aufweist um Bindungen der Lithium-Ionen zu verhindern. Beim Anlegen einer Spannung an den beiden Kontakten der Elektroden wird der Vorgang umgekehrt und die Potentialdifferenz an den Aktivmaterialien wieder hergestellt. Um die erwähnte Ladungsneutralität zu erhalten, wandern auch die Lithium-Ionen wieder durch den Elektrolyt in die nun entgegengesetzte Richtung und lagern sich an der positiv geladenen Elektrode ein, was einem Ladevorgang entspricht [JW06].
Der Elektrolyt besteht üblicherweise aus nicht wässrigen, aprotischen Lösungsmitteln mit zusätzlichem Leitsalz für die ionische Leitfähigkeit. Im Gegensatz dazu können auch polymere, gelartige Elektrolyte verwendet werden, die zwar eine geringere Leitfähigkeit aufweisen, aber als auslaufsicherer gelten und deshalb flexibler verbaut werden können.
Die Alterung von Lithium-Ionen Zellen wirkt sich in Kapazitäts- und Leistungsreduktion aus und wird von zwei Faktoren bestimmt. Der zyklischen und der kalendarischen Alterung. Zyklische Alterung wird in erster Linie durch Deckschichten verursacht, die sich zwischen der negativen Elektrode und dem Elektrolyt bildet. Verursacht werden sie z.B. durch Zersetzungsprodukte im organischen Elektrolyt.

Nicht zu verwechseln mit der *Solid Electrolyte Interphase* (SEI), einer irreversible Deckschicht, die sich beim ersten Ladevorgang aufbaut, um als für Ionen durchlässiger Korrosionsschutz zu wirken [Bot12].
Einen weiteren Einfluss auf die zyklische Alterung verursacht die Volumenänderung der Aktivmaterialien, z.B. bei Graphit. Dabei ändert sich während des Lade- oder Entladevorganges durch die Veränderung der Einlagerungsmenge der Lithium-Ionen die räumliche Ausdehnung des Aktivmaterials, weshalb die Zellhüllen und die Kontaktflächen darin mechanisch belastet werden. Kalendarische Alterung nennt man den unvermeidbaren Kapazitäts- und Leistungsverlust der Zelle, für den es mehrere Ursachen gibt. Zum einen werden gewisse Anteile in den Aktivmaterialien mit der Zeit abgebaut und damit die Aufnahmefähigkeit von Lithium-Ionen reduziert. Zum Anderen können Verunreinigungen im Elektrolyt Lithium-Ionen dauerhaft ins Aktivmaterial binden [JW06]. Eine Überschreitung der Spannungsgrenzen hat immer einen irreversiblen Kapazitätsverlust zur Folge, der mit einer Beschädigung der Zelle gleichzusetzen ist. In diesen Fällen werden chemische Zersetzungs- und Korrosionsprozesse beschleunigt, welche den Innenwiderstand der Zelle erhöhen. In der Praxis werden Zellen typischerweise bis 80% ihrer ursprünglichen Kapazität verwendet, danach entsorgt und recycelt [Kle09].
Der Spannungsverlauf über den Ladezustand von 0-100% weist bei Messungen realer Batterien ein nichtlineares Verhalten auf (siehe Abb. 1.9). Anhand des Beispiels, der in dieser Arbeit modellierten Zelle mit einer Lithium-Eisenphosphat Anode (siehe Abschnitt 1.5.1), ist im unteren Spannungsbereich bzw. Ladezustand eine Abflachung des Anstieges der Zellspannung erkennbar. Sie wird durch die Abnahme der Lithium-Transfergeschwindigkeit beim Übergang vom Aktivmaterial in den Elektrolyt verursacht. Der mittlere Spannungs- bzw. Ladezustandsbereich nähert sich einem linearen Verlauf an, der von Strom und Innenwiderstand abhängig ist. Trotzdem sind in ((siehe Abb. 1.9), rote Kreise) unregelmäßige Spannungshübe erkennbar, in [JW06] Potentialhübe genannt.
Deren Ursache liegt im Diffusionsverhalten von Lithium-Ionen bei der Einlagerung in Eisenphosphat. Dabei werden die Lithium-Ionen in bestimmter Reihenfolge in das tunnelartig aufgebaute Lithium-Eisenphosphat eingelagert. Die fortschreitende Einlagerung führt zu unterschiedlichen Sättigungsgraden des Aktivmaterials, welche bei Überwindung in unregelmäßigen Potentialanstiegen zu erkennen sind [JW06]. Im oberen Spannungsbereich bzw. Ladezustand ist abschließend ein steiler Anstieg der Zellspannung erkennbar, da, vereinfacht betrachtet, Lithium-Ionen in diesem Bereich wenige freie Stellen zur Diffusion finden. Dasselbe Verhalten ist erkennbar, wenn Lithium-Ionen in die entgegengesetzte Richtung wandern. Jedoch treten an den Übergängen zwischen Aktivmaterialien und dem Elektrolyt andere Transferge-

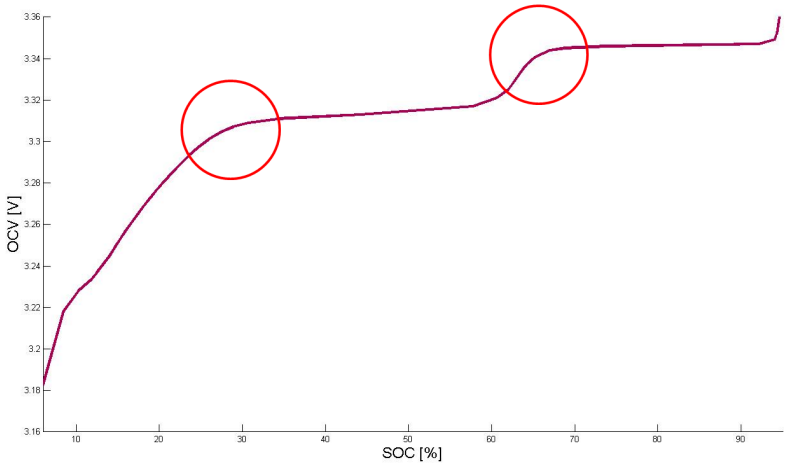

Abbildung 1.9: Zusammenhang zwischen der Ruhespannung (OCV, (siehe Abschnitt 1.5.3.4)) und dem Ladezustand (SOC, (siehe Abschnitt 1.5.3.1))

schwindigkeiten und unterschiedliche Geschwindigkeiten von Ein- und Auslagerung der Lithium-Ionen auf. Dieses Verhalten wird bei Messungen in Form von unterschiedlichen Spannungsverläufen von Lade- und Entladevorgang erkennbar, die eine Hysterese-Funktion der Spannung bilden (siehe Abb. 1.10).

Ein wichtiges Merkmal im Zellverhalten bietet das Relaxationsverhalten am Ende einer Strombelastung. Vereinfacht erklärt handelt es sich dabei um Stauungen an den Übergängen zwischen dem Aktivmaterial und dem Elektrolyt in Lade- und Entladerichtung, da Diffusionsvorgänge zuerst an der Oberfläche des Aktivmaterials und erst nach Mangel an freien Einlagerungsplätzen weiter hinten stattfinden. Sie verursachen eine verzögerte Einlagerung des Lithium-Ions ins Aktivmaterial, einen Stau an der Grenzschicht. Dieser Effekt beschreibt die Spannungsantwort der Zelle. Sie variiert mit der Höhe des Belastungsstromes und der Betriebstemperatur, weil dadurch die Lithium-Ionen unterschiedliche kinetische Energien erhalten, die sich im Maß dieser Stauung auswirkt [JW06].

Wiederum in Bezug auf den speziellen Fall der Lithium-Eisenphosphat Anode in Verbindung mit einer Graphit Kathode (siehe Abschnitt 1.5.1) liegt die optimale Betriebstemperatur zwischen 0 und 30° Celsius, welcher auf gemäßigte Klimazonen optimiert wurde. Oberhalb von 30° hat sie einen negativen Einfluss auf die Lebensdauer. Dabei wird der Elektrolyt zersetzt, wobei dieser Effekt erst bei sehr hohen Betriebstemperaturen merkbaren Einfluss hat. Im Betrieb bei 0° Celsius und darunter sinkt die Diffusionsgeschwindigkeit der Lithium-Ionen innerhalb der Zel-

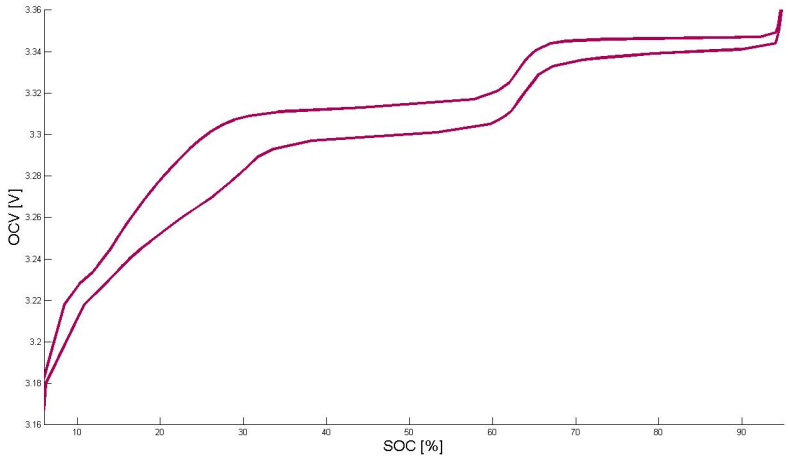

Abbildung 1.10: Unterschied im Ruhespannungsverlauf (Hysterese) im Lade- und Entladefall

le. Eine Art Trägheit der Ein- und Auslagerungsvorgänge wird erkennbar (siehe Abb. 1.11). Speziell bei der Verlangsamung der Einlagerungs-Kinetik an der Graphit Anode, können unregelmäßige Ablagerung von metallischem Lithium stattfinden und Dendriten in Richtung Kathode wachsen lassen[1], dem sogenannten Lithium Plating. Verursacht wird dieser Vorgang durch hohe Lade-/Entladeströme bei niedrigen Betriebstemperaturen. Im Betrieb innerhalb der definierten Betriebsgrenzen sollte hingegen kein metallisches Lithium entstehen. Im schlimmsten Fall wachsen die Dendriten durch den Elektrolyt und den Separator zur zweiten Elektrode. Der dabei entstehende, interne Kurzschluss führt zur Zerstörung der Zelle [Ele12]. Zu hohe Betriebsströme bilden generell eine Gefahr für die Zelle, da sie zum einen die Eigenerwärmung und damit die Betriebstemperatur drastisch erhöhen, materialabhängig auch den Innenwiderstand erheblich anheben. Zum anderen zerstören sie die SEI und beeinflussen damit das Verhalten der Zelle negativ. Die Anschlusskontakte der Elektrode können ebenfalls dadurch zerstört werden [JW06].
Die beschriebenen Eigenschaften und Effekte der Lithium-Ionen Technologie bieten in automotiven Hochstrom-Anwendungen eine Reihe von Vorteile, weshalb sie alternativen Batterietechnologien mittlerweile vorgezogen wird. [Bot12] betont den hochreversiblen Einlagerungsprozess von Lithium-Ionen von mehr als 1000 Lade- bzw. Entladezyklen und die höchsten Werte der spezifischen Energie solcher Systeme im Vergleich mit chemischen Systemen auf Nickel-Basis (Vergleich siehe Tabelle

[1]Dendritenwachstum: Die entstandene, metallische Schicht auf der Elektrode verändert das Auslagerungs-Potential von Lithium, was zu Veränderungen im Ruhespannungsverlauf über den Ladezustand führt und in irreversiblem Kapazitätsverlust resultiert [BS10]

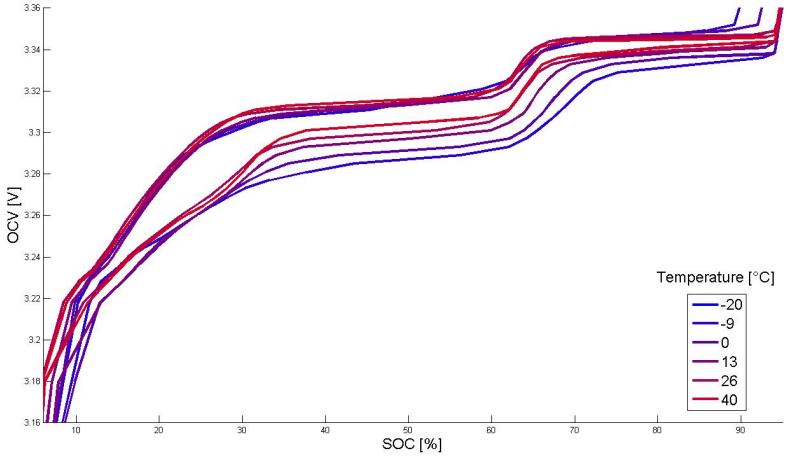

Abbildung 1.11: Unterschied im Ruhespannungsverlauf (Hysterese) im Lade- und Entladefall (im Temperaturbereich von -20° bis 40° Celsius)

1.3.7).
Das angedeutete, nichtlineare Spannungsverhalten, die starke Temperaturabhängigkeit und der Sicherheitsaspekt bei solch hohen Energiedichten stellt die Forschung und Entwicklung aber weiterhin vor neue Aufgaben, wie im Abschnitt 1.6 gezeigt wird.

1.5 Das Batteriesystem

Das folgende Kapitel bietet einen Einblick in den Aufbau und die Funktionsweise des in dieser Arbeit verwendeten PHEV-Batteriesystems. Nach einer kurzen Erläuterung der Hardwarekomponenten wird der Schwerpunkt in diesem Abschnitt auf die Funktion der Steuerungssoftware gelegt.

1.5.1 Beschreibung der Hardwarekomponenten des Batteriesystems

Die Kapazität und die Leistungsfähigkeit der Batterie werden aus einem Verband von mehr als 100 in Serie geschalteten Zellen bestimmt. Die Zelle beinhaltet eine Lithium-Eisenphosphat Anode, eine Graphit Kathode und einen für solche Anwendungen standardisierten Elektrolyten aus einer Lithium-Phosphor-Flur Salzlösung (siehe Abschnitt 1.4). Die Kapazität einer einzelnen Zelle beträgt laut Datenblatt $21Ah$. Bedingt durch die Serienschaltung wird das gesamte System ebenfalls mit

einer Nennkapazität von $21Ah$ beziffert, aus Testerfahrung sollten aber mit realen $20Ah$ gerechnet werden. Der gesamte Zellverband liefert bei Raumtemperatur eine maximale Entladeleistung von etwa $90kW$.
Neben den galvanischen Zellen besteht die Batterie aus einer Reihe weiterer elektronischer Komponenten. Die bereits erwähnte BMU (siehe Abschnitt 1.3.4) bildet dabei das Rechenwerk, auf dem die Steuerungssoftware betrieben wird. Sie stellt auch die Schnittstellen zur Kommunikation mit anderen Fahrzeugkomponenten zur Verfügung, z.B. dem *Controller Area Network* (CAN-Bus) oder die Verbindung zur *Vehicle Control Unit* (VCU).
Zum Schutz des Systems existiert eine Relais-Steuerung, zur Ausgrenzung hoher Einschaltströme per Precharge-Stromkreis, sowie ein Isolationswächter und eine Reihe von High-Voltage Gleichstromsicherungen. Die Zelltemperatur reguliert sich mit flüssigem Kühlmittel, welches per Leitungssystem von einer externen Quelle durch das System gepumpt wird (siehe Abschnitt 1.3.5). Weitere Sicherheitsmaßnahmen sind in Form von Abschaltmechanismen bei zu hohen Systemspannungen (*Hazardous/High Voltage Interlock Loop*, HVIL) integriert.
Die Messung für Spannung und Temperatur, übernimmt für kleinere Zellverbände je ein Elektronikmodul, das *Cell Supervision Circuit* (CSC). Den Belastungsstrom über den Zellverband misst eine Kombination aus zwei Sensoren, die aus Hall-Sensor und einer Shunt-Messung besteht.
Gemeinsam bilden die beschriebenen Komponenten das *Energy Storage System* (ESS), dessen Blockaufbau in (siehe Abb. 1.12) zu sehen ist [MSB12].

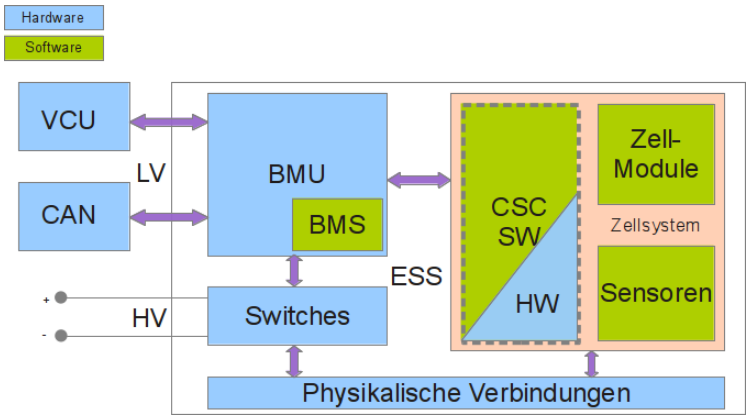

Abbildung 1.12: Blockaufbau des Energy Storage System

Für den korrekten Startvorgang des BMS müssen manche Werte aus dem *Non Volatile Memory* (NVM) ausgelesen werden, welches sowohl EEPROM als auch Flash-Speicher verwendet. Die Kombination der Speicherelemente wird aus Lebensdauer- und Kostengründen integriert, da einige Signale im Batteriebetrieb sehr oft ausgelesen werden müssen, andere nur selten, z.B. wird nur bei Systemstart die Belastungsvergangenheit der Batterie gelesen. Des Weiteren werden Informationen des letzten berechneten Ladezustands (siehe Abschnitt 1.5.3.1), oder Informationen die den Alterungszustand beschreiben (siehe Abschnitt 1.5.3.3), aus dem NVM geladen.

1.5.2 Beschreibung der Softwarefunktionalität des Batteriesystems

Die Aufgaben des BMS strecken sich über mehrere Funktionsbereiche. Es dient im Allgemeinen zur Berechnung und Überprüfung von diversen Signallimitierungen, oder zur Bestimmung von Zustandsgrößen, die dem Fahrzeuglenker Informationen und Rückmeldungen über sein Fahrverhalten im Elektro- und Hybridmodus liefern [Kle09]. Beim Betrachten der einzelnen Module des BMS wird die Aufteilung der Funktionalität ersichtlich. Die Rechenalgorithmen unterteilen sich folgendermaßen in jene:

- der Statussignale (Status Calulation)
- des Ladezustands (*State of Charge*, SOC)
- der Leistungslimitierungen (*State of Power*, SOP)
- des Zell-Balancings (CellBalancing)
- der Kapazitätsberechnung (CapaDet)
- der Alterungsberechnung (*State of Health*, SOH)
- des spannungsbasierten Zellverhaltens (*Voltage Cell Modell*, VCM)

Die Grundlage der Berechnungen bilden Eingangssignale physikalischer Größen und Werte verschiedener Zustände. Die Zelle, oder wie im Falle der Simulationsumgebung, das *Zellmodell* (ZM), liefert die physikalischen Informationen in Form von Strom-, Spannungs- und der Temperaturmessung. Die genaue Funktion des ZMs wird in Abschnitt 2.1.4.1 vorgestellt.

1.5.3 Beschreibung der Softwarefunktionen

Die Bestimmung von Batteriezuständen und die Ermittlung ihrer Charakteristiken basiert auf teilweise komplexen Algorithmen. Die einzelnen Funktionen werden im Folgenden näher erläutert.

1.5.3.1 SOC

Der Ladezustand der Batterie stellt eine der wichtigsten Zustandsgrößen des BMS dar, da er als bestimmende Größe für die Reichweite des Elektroautos gilt. Abhängig ist seine Berechnung vom Systemstrom, der Zell-/Systemspannung, der Systemtemperatur, den Batterieinnenwiderständen und dem gemessenen Wert der *Open Circuit Voltage* (OCV) (siehe Abschnitt 1.5.4.2). Letztere ist eine sehr wichtige Information, da wie in (siehe Abschnitt 1.4) beschrieben, die Zelle in Lade- und Entladerichtung unterschiedliche Spannungskennlinien in Abhängigkeit des Ladezustands aufweist. Demnach muss bei der Berechnung des SOCs der Einfluss von vergangenen Zellbelastungsprofilen berücksichtigt werden. Im flachen Bereich des Spannungsbereichs kann im Falle einer falschen Hysterese-Information, in Verbindung mit der Genauigkeit der gemessenen Spannung, ein Fehler von bis zu ca. 40% SOC entstehen. Die Bestimmung des Ladezustands und weiterer Zustandsgrößen des Batteriesystems erfolgt anhand des niedrigsten (`SocMin`), des höchsten (`SocMax`) und des Durchschnittswerts über alle Zell-SOCs (siehe Abschnitt 1.5.4.3).

1.5.3.2 SOP

Um das Vermögen der Leistungsaufnahme und -abgabe der Batterie für die unmittelbare Zukunft schätzen zu können, werden die SOP-Signale berechnet. In Abhängigkeit vom geforderten Systemstrom, den Systemspannungsgrenzen, dem Batterieinnenwiderstand und den Zelltemperaturen, sind sie dafür verantwortlich, Systemstrom, -spannung und –leistung, den Umständen entsprechend, in Höhe und Zeit zu begrenzen, um eine dauerhafte Beschädigung der Batterie zu verhindern. Unter diesen Signalen findet man auch jene, welche die Zellen des Batteriesystems vor dauerhaften Schäden bei der Über- oder Unterschreitung der Systemspannungsgrenzen bewahrt.
Dafür muss der Algorithmus auch die vergangene Belastungsgeschichte des Systems kennen. Die Ausgangssignale der SOP-Prognose berücksichtigen zwei Zeitspannen für die bevorstehende Belastung. Die wenige Sekunden andauernde, kurze Zeitspanne sichert das System gegen kurzzeitige, aber energiereiche Belastungsspitzen ab.

Die lange Zeitspanne erstreckt sich über wenige Minuten und soll das Leistungsvermögen der Batterie bei eher leistungsarmen Lade- oder Entladevorgängen, z.B. an der Ladestation, prognostizieren [MSB12].

1.5.3.3 SOH

Der SOH, welcher die Zellalterung bestimmt, befindet sich im B-Muster-BMS (siehe Abschnitt 1.3.6) noch in der Entwicklungsphase. Die Funktion basiert auf einem Zeitmodell, welches die Zellbelastungsprofile der Vergangenheit protokolliert und in Abhängigkeit vom Batteriealter die Leistungsfähigkeit sowie die nutzbare Energie begrenzt. Das Alter liefert eine Zustandsgröße zwischen Startzeitpunkt (*Begin of Life*, BOL) und Endzeitpunkt (*End of Life*, EOL).

1.5.3.4 OCV

Der Ruhespannungspegel wird aus einem der Zelle äquivalenten Spannungsersatzschaltbild im OCV-Modul berechnet (siehe Abschnitt 1.5.4.2) und hängt von der Temperatur und dem SOC ab. Er bildet gemeinsam mit dem SOC den nichtlinearen Zusammenhang des Ladezustandes mit der Ruhespannung, der als charakteristisch für die Li-Ion Technologie gilt und deshalb häufig in der Literatur behandelt wird (siehe Abschnitt 1.4 und 1.6). Ebenso wie beim SOC (siehe Abschnitt 1.5.3.1) werden die Zellspannungen anhand des niedrigsten (`OcvMin`), des höchsten (`OcvMax`) und des durchschnittlichen Spannungspegels dargestellt (siehe Abschnitt 1.5.4.2). Abschnitt 1.4 beschreibt, warum sich die Lade- und Entladeverhalten unterscheiden und in Abhängigkeit des SOCs eine Hysterese bilden.

1.5.3.5 Coulomb Counting

Bei dieser Berechnung wird ein Start-SOC-Wert für die Initialisierung benötigt. Der Belastungsstrom wird während des Betriebes in Lade- und Entladerichtung über die Zeit integriert und bietet somit die Information der Ladezustandsveränderung der Batterie. Unter der Voraussetzung, dass die momentane Kapazität der Batterie bekannt ist, kann damit der aktuelle, strombasierte Ladezustand berechnet werden. Aufgrund von Ungenauigkeiten der Stromsensoren ist nach längerer Betriebszeit mit ungenauen Messergebnissen zu rechnen, weshalb eine Rekalibrierung des SOCs notwendig wird.

1.5.3.6 Methode zur Darstellung der Signalfehler

Im Batteriemodell hat ein Signal, dem Messunsicherheiten zu Grunde liegen, eine zusätzliche Information dieses Fehlers als eigenes Signal. Genau genommen stellen zwei Fehlersignale einer Messgröße die maximale Abweichung für den Fall der steigenden (positiven) und den Fall der fallenden (negativen) Messgröße dar. So werden bspw. bei der Berechnung der SOC-Signale (`SocMin` und `SocMax`) zu jedem Zeitpunkt die positiven und negativen Fehler der gemessenen Strom-, Spannungs- und Temperaturwerte mitberücksichtigt. Graphisch dargestellt spannen sie einen Signalbereich auf, der einen Eindruck auf die Korrektheit des Signals liefert, wobei die Bereichsgrenzen den „worst-case“ der Signalfehlergrenzen darstellen (siehe Abb. 1.13). In der Regel werden strombasierte Signalfehler stetig erhöht, bis eine Korrektur des Basissignals durchgeführt wird und damit auch der dazugehörige Fehler reduziert werden kann. Grund für die Aufteilung der Signalfehler in eine positive und eine negative Richtung ist die zu erwartende Asymmetrie der Fehlerentwicklung, die für die Signal- und Systemanalyse transparent gehalten werden muss.

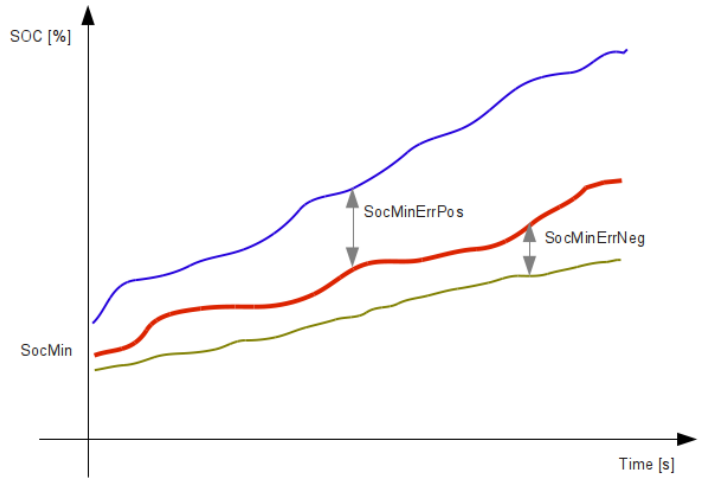

Abbildung 1.13: Mit fortlaufender Zeit wachsen die beiden Signalfehler (`Neg` und `Pos`), hier am Bsp. des `SocMin`, in positive Richtung an

1.5.3.7 Relaxation

Der Relaxationseffekt beschreibt die Spannungsantwort der Zelle nach Betragsänderung der anliegenden Last. Der Grund für diesen Effekt und die Form seiner Auswirkung ist mit dem Diffusionsverhalten der Li-Ion-Technologie zu erklären (siehe Abschnitt 1.4). Die Form der Spannungsantwort ist in ((siehe Abb. 1.14), schematische

Darstellung) zu sehen. Sie setzt sich aus dem Spannungsabfall am Innenwiderstand und jenen an den Polarisationswiderständen zusammen ($U_{Ri} + U_{pol}$). Wird die Zelle entlastet, verbleibt nach dem Spannungseinfluss an den Widerständen ($U_{Ri} + U_{rel}$), die um ΔOCV angehobene Zellspannung (siehe Abb. 1.14).

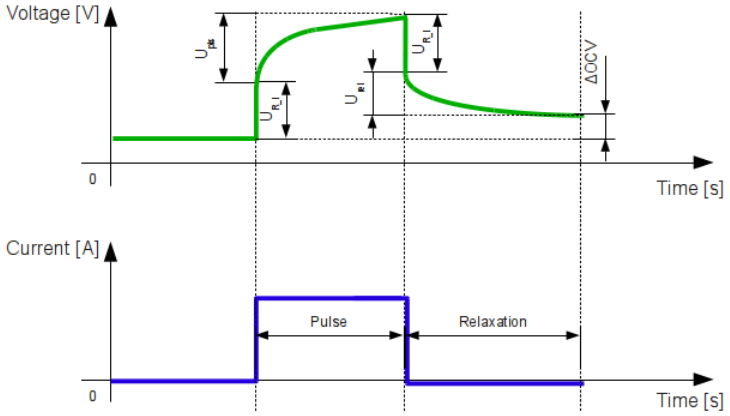

Abbildung 1.14: Darstellung des Relaxationsverhaltens anhand des Spannungsverlaufs, der aus einem Strompuls resultiert

$$U_{pol} = [R_i + R_1(1 - \mathrm{e}^{\frac{-t}{\tau_{pol1}}}) + R_2(1 - \mathrm{e}^{\frac{-t}{\tau_{pol2}}})] * I \tag{1.1}$$

$$U_{rel} = U_{pol1}\mathrm{e}^{\frac{-t}{\tau_{rel1}}} + U_{pol2}\mathrm{e}^{\frac{-t}{\tau_{rel2}}} \tag{1.2}$$

Parameter:

I	Belastungsstrom
R_i	Innenwiderstand
R_1	Widerstandswert des ersten RC-Gliedes
R_2	Widerstandswert des zweiten RC-Gliedes
τ_{pol1}, τ_{pol2}	Polarisations-Zeitkonstanten des ersten bzw. zweiten RC-Glieds, (R_1C_1 bzw. R_2C_2)
τ_{rel1}, τ_{rel2}	Relaxations-Zeitkonstanten des ersten bzw. zweiten RC-Glieds, (R_1C_1 bzw. R_2C_2)

Das Impedanz-Ersatzschaltbild, an denen die beschriebenen Spannungsabfälle entstehen, ist in (siehe Abb. 1.22) zu sehen. Der Polarisations- bzw. der Relaxationswiderstand setzt sich aus der Charakteristik der beiden seriellen RC-Glieder zusammen. Die sinnvolle Anzahl der RC-Glieder wird wie in [ZC10] oft diskutiert und variiert [Zha10].

1.5.3.8 Balancing

Der Begriff Balancing beschreibt das Verfahren, welches die Zellspannungen in einem seriellen Zellverbund auf dasselbe Potential bringt. Dieser Schritt ist notwendig, da die Zellen aufgrund von Fertigungstoleranzen leichte Unterschiede in ihren chemischen Charakteristiken aufweisen. Nach längerer Belastung der Batterie, also einem häufigen Richtungswechsel des Belastungsstromes über den Zellverbund, werden sich erwartungsgemäß unterschiedliche Ladungszustände in den Zellen einstellen. In weiterer Folge wird dadurch Leistungs- und Energieverlust im Batteriesystem zu erwarten sein, da die Kapazität der gesamten Batterie von der Kapazität der schwächsten Zelle bestimmt wird [HLS11]. Der Algorithmus sollte bevorzugt im ruhenden System, bei hoher SOC-Spreizung im Zellverbund, aber auch im Betrieb unter bestimmten Kriterien (siehe Abschnitt 1.5.3.9), ausgeführt werden. Dafür sollte aber für jede einzelne Zelle die Kapazität bekannt sein.

1.5.3.9 Kapazitätsbestimmung

Die aktuelle Kapazität einer Zelle stellt eine wichtige Zustandsgröße zur Bestimmung des Alters, des Ladezustandes oder eines möglichen Defektes dar. Der Algorithmus basiert auf einem Entladevorgang der Zelle, wobei die Temperatur dabei konstant gehalten werden sollte [HLS11]. Dafür muss die Stromhöhe entsprechend eingestellt werden (z.B. 1C [2]), um eine zu hohe Selbsterwärmung des Systems zu vermeiden. Davor sollte sich die Zelle eine Zeit lang in Ruhe befunden haben, da die Messung von der Relaxation nicht beeinflusst werden darf. In Abhängigkeit der Stromhöhe kann mittels Zeitdauer bis zum Erreichen des unteren Zellspannungslimit die Kapazität errechnet werden. Wiederrum stellt die Relaxation der Zelle die Bedingung, den Strom, kurz vor Ende des Entladevorganges, stetig zu reduzieren, da mit hohen Strömen die tatsächliche Spannungsgrenze nicht erreicht werden kann [HLS11] (siehe Abb. 1.15).

[2] 1C entspricht dem Betrag des Konstantstromes, der notwendig ist, um die darauf bezogene, anfangs entladene Zelle, innerhalb einer Stunde komplett zu laden

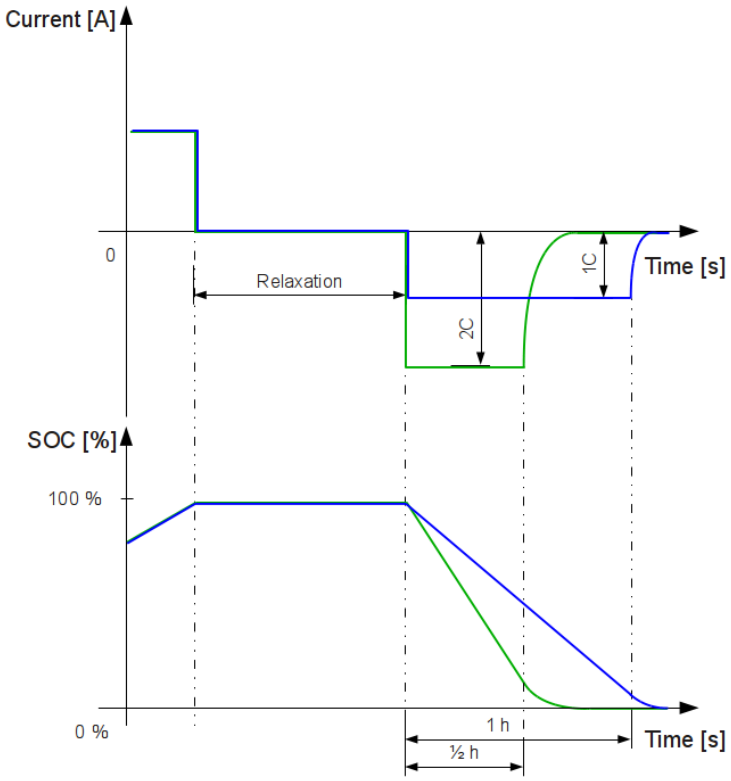

Abbildung 1.15: Vereinfachte Darstellung der Kapazitätsbestimmung anhand von zwei unterschiedlichen Konstantströmen (1C, 2C)

1.5.3.10 Zellcharakterisierung

Die Zellcharakterisierung dient der Wertedefinition der Zellparameter. Die Zellparameter beschreiben das Spannungs-, das Widerstands- und das Alterungsverhalten der Zelle über unterschiedliche Belastungszustände, in realistischen Temperaturbereichen, bei jedem Ladungszustand.
Dabei wird für die OCV-Bestimmung ein SOC-Bereich mit pulsförmigen Eingangsströmen durchschritten und nach jedem Puls die Zellspannung gemessen [ZC10]. Zu beachten ist, dass nach jedem Strompuls aufgrund des Relaxationsverhaltens der Zelle eine gewisse Zeit abgewartet werden muss, um unbeeinflusste Werte, in diesem Fall die OCV, messen zu können.

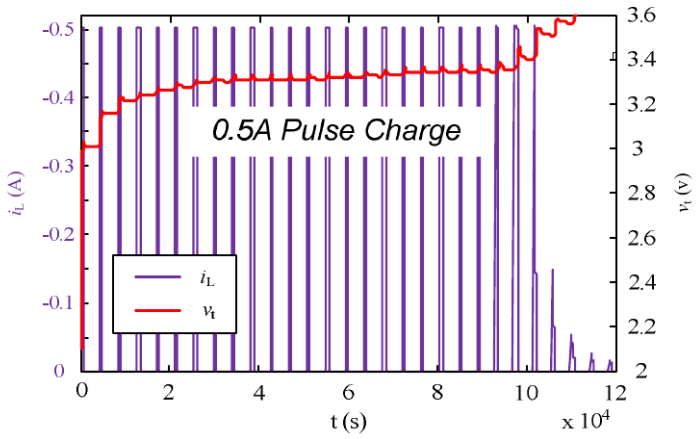

Abbildung 1.16: OCV-Bestimmung mittels Strompulsen aus [ZC10]. Genauere Beschreibung dieses Experiments (siehe Abschnitt 1.6.4)

Die Länge dieser Pausen beruht auf Erfahrungswerten, da generell mit mehreren Stunden gerechnet werden muss, bis sich eine Zelle von einer Strombelastung erholen kann, vor allem bei tiefen Umgebungstemperaturen. Da diese Zeitspannen in der Entwicklung nicht praxistauglich sind, wird an der Zelle in kurzen Abständen die Spannungsänderung während der Relaxation gemessen (ΔOCV [mV]), bis diese ausreichend klein ist, um die Zelle in einem angenäherten Ruhezustand betrachten zu können. In der Praxis wird die OCV bei Raumtemperatur nach einer Stunde bestimmt, im Tieftemperaturbereich (unter 0° C) erhöht sich die Wartezeit auf bis zu drei Stunden (siehe Abschnitt 1.4).
Beginnend bei einem Start-SOC-Wert und einer definierten Temperatur wird die Zelle mit definierten Strompulsen belastet, bis entweder die bautechnisch bedingte oder die definierte Maximalspannung erreicht wird. Vorteil dieser Methode zur Gewinnung der Spannungscharakteristik ist die hohe Genauigkeit, den SOC daraus abbilden zu können. Nachteilig wirken sich vor allem die lange Zeitdauer des Verfahrens und der benötigte Speicherplatz der Parameter am System, sowie der Einfluss der Messungenauigkeiten am Testsystem aus.
Ein weiteres Verfahren zur Charakterisierung einer Zelle dient der Identifikation des Energiedurchsatzes, um den in (siehe Abschnitt 1.5.3.1) erwähnten Unterschied zwischen der Lade- und der Entladetrajektorie der Spannungscharakteristik zu überwinden. Im sogenannten Hysterese-Test (siehe Abschnitt 1.5.4.2) wird unter definierten Betriebsbedingungen der Wechsel zwischen den beiden Trajektorien gemessen.
Im Resistance-Test wird die Zelle mit variierenden, pulsförmigen Strömen belastet, um ein Puls- und Relaxationsverhalten (siehe Abschnitt 1.5.3.7) für verschiedene

Belastungsfälle messen zu können. Zur Modellierung dieses Effekts werden die simulierten Spannungsantworten des ZMs per Fitting (siehe Abschnitt 1.5.3.11) an die gemessenen Spannungsverläufe bestmöglich angepasst.
Die in (siehe Abschnitt 1.5.3.2) erwähnten Stromlimits erhält man mit Hilfe des Maximalstrom-Belastungs-Tests. Dieser Test belastet die Zelle an einem bestimmten Start-SOC-Wert bspw. mit einem eher kurzen, dafür hohen Ladestrom. Gemessen wird dabei die Tatsache, ob dieser Strompuls überhaupt geladen werden kann ohne die obere Zellspannung zu erreichen, bzw. die Dauer, wann in solch einem Fall die Spannungsgrenze erreicht wird. Danach reduziert sich der Strom pulsartig auf ein Minimum (siehe Abschnitt 2.4). Dasselbe Verfahren findet auch für langzeitliche, kleinere Belastungsströme statt, beides auch für den Fall der Zellentladung. Somit wird die mögliche Zellbelastung bei kurzzeitigen und langzeitigen Strompulsen, in Abhängigkeit vom SOC und vom Zell-Innenwiderstand, generiert (siehe Abschnitt 1.5.3.2).

1.5.3.11 Fitting-Algorithmus

Um im Modell den zeitlichen Verlauf der Spannungsantwort nach einer Zellbelastung an das reale Vorbild angleichen zu können, müssen entsprechende Parameter des BMS, welche dessen Ersatzschaltbild im VCM bedaten (siehe Abschnitt 1.5.4.2), gefunden werden. Konkret handelt es sich dabei um die Widerstandswerte (R [Ω]), die Zeitkonstanten (τ [s]) der RC-Glieder und den Wert des Innenwiderstandes (R_i [Ω]) (vergleiche Relaxation in Abschnitt 1.5.3.7). Dafür wird ein Algorithmus angewendet, der die Parameter der darauf aufbauenden Systembeschreibungsgleichungen so lange anpasst, bis deren Ergebnis dem am Prüfstand gemessen Spannungsverlauf bestmöglich gleicht (siehe Abschnitt 1.5.3.11).
Dafür wird die Spannungsantwort der Zelle in zwei Teile getrennt, den sogenannten Puls- und den Relaxationsteil. Der Pulsteil zeigt den Zellspannungsverlauf nach Aufschaltung einer Belastung, der wegen seiner hohen Dynamik sehr schnell abgetastet wird (wenige Zehntel-Sekunden Sample-Time). Der Relaxationsteil zeigt das Abklingen der Zellspannung nach Abschalten der Belastung und wird wegen seines trägen Verhaltes eher langsam (wenige Sekunden Sample-Time) abgetastet (siehe Abb. 1.17). Die Dauer der Aufzeichnung hängt dabei von der Höhe der Zell-Belastung ab. Je nach Test wird die Zelle an verschiedenen Ladezuständen mit mehreren Strompulsen belastet, wobei jede Spannungsantwort für das Fitting einzeln betrachtet wird. Anschließend wird für jeden dieser Teile der Fitting-Algorithmus angewendet, der unter der Einschränkung des nachgebildeten Systems zweiter Ord-

nung (siehe Abschnitt 1.5.4.2), den Spannungsverlauf mit Hilfe von fünf Parametern (siehe Abschnitt 1.5.3.7) anpasst.

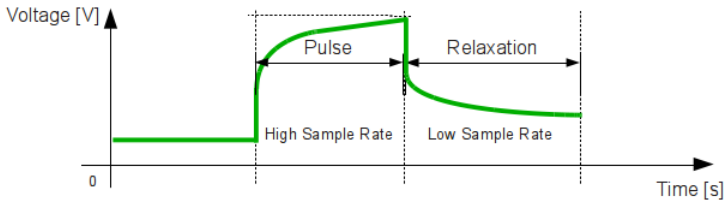

Abbildung 1.17: Aufteilung der Spannungsantwort in Puls- und Relaxationsteil

Beim Pulsteil kann der Einfluss von vorangegangenen Pulsen in der Praxis nicht beseitigt werden. Deswegen wird der Signalverlauf vor t_1 des Spannungsverlaufs in Abb. 1.19 bei der Bewertung des Fitting-Ergebnisses nicht mitberücksichtigt, weil in dieser Zeitspanne Messfehler auftreten können. Da der Algorithmus außerdem auf den weniger dynamisch wirkenden Spannungsverlauf ab t_2 optimiert wurde, macht sich diese Maßnahme kaum im Ergebnis bemerkbar (t_3 wird in diesem Rahmen nicht verwendet).

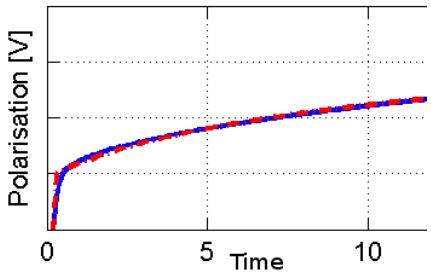

Abbildung 1.18: Bsp. Fitting des Polarisationsverhaltens: (blau gemessen, rot gefittet))

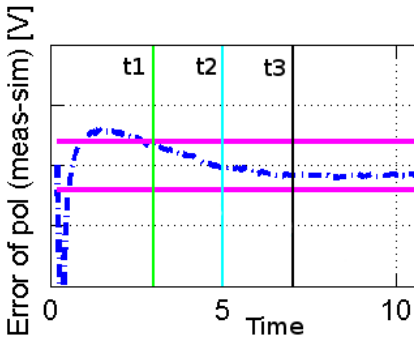

Abbildung 1.19: Bsp. Abweichung zw. gemessener und gefitteter Spannung

Aus (siehe Abb. 1.18) wird erkennbar, dass für Spannungsverläufe höherer Ordnung, die mittels System 2. Ordnung angenäherte Linie (strichliert), dem gemessenen Spannungsverlauf (blau) in diesem Fall noch gut folgen kann. Das Ergebnis des Algorithmus (siehe Abb. 1.19) gilt als gut, wenn sich der quadratische Fehler (siehe Gleichung 1.5.3.11) in positiver und negativer Richtung noch innerhalb akzeptabler Grenzen befindet (rosa Linie), darf aber in bestimmten Situationen, z.B. wenn ein hoher Belastungspuls sehr lange andauert, diese geringfügig überschreiten, um dennoch akzeptiert zu werden.

$$Err = \sum[(U_{fit} - U_{meas})^2] \tag{1.3}$$

Err	Quadratische Fehler zw. U_{fit} und U_{mesh}
U_{fit}	Berechnete Spannung (rot in Abb. 1.18)
U_{meas}	Gemessene Spannung (blau in Abb. 1.18)

Die Parameterlisten über alle Spannungsantworten, jeweils für den Puls- und den Relaxationsteil, werden in einem weiteren Arbeitsschritt auf Plausibilität verglichen, um fehlerhafte Ergebnisse identifizieren und korrigieren zu können.

1.5.4 Das BMS-Modell

Das BMS setzt sich aus einer Reihe von Funktionsmodulen zusammen und bietet die Kommunikationsschnittstellen zu den weiteren, externen Batteriekomponenten. Die vorher vorgestellten Funktionen werden von diesen Modulen implementiert und im Folgenden detailliert beschrieben.

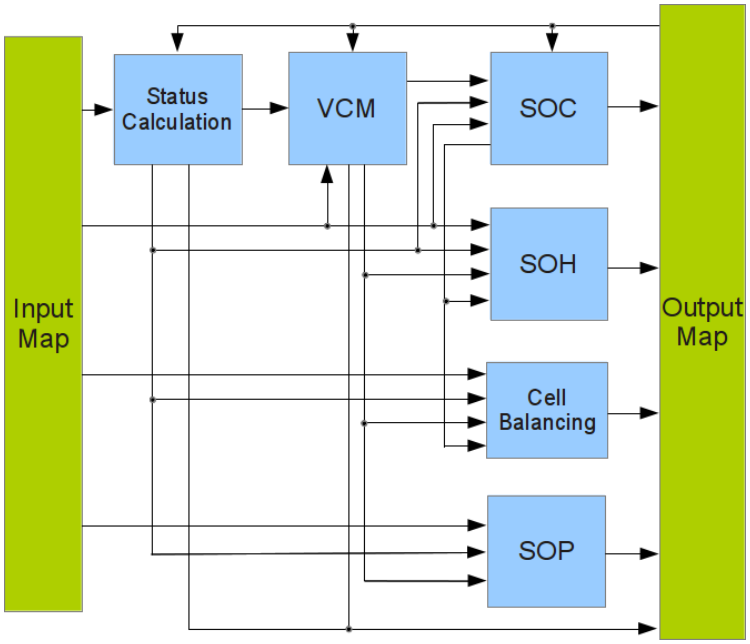

Abbildung 1.20: Blockaufbau des BMS

1.5.4.1 Status Calculation

Die Aufgabe dieses Moduls ist es, den erfolgreichen Batteriestart mit Statusflags zu signalisieren. Damit wird der korrekte Funktionsverlauf bei Systemstart überprüft und in weiterer Folge die Initialisierung per NVM freigeben. Als Input erhält dieses Modul die Statussignale der Messsensoren von Eingangsstrom, -spannung und –temperatur sowie den NVM-Status.

1.5.4.2 VCM

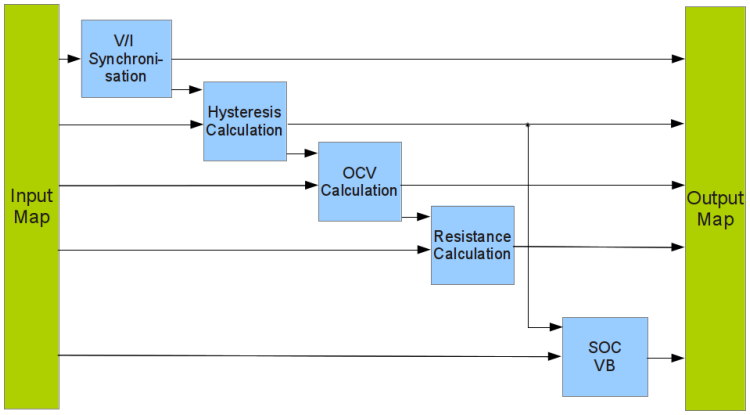

Abbildung 1.21: Blockaufbau des VCM

Das *Voltage Cell Modell* (VCM) modelliert das galvanische Verhalten der im System integrierten Batteriezellen, nicht zu verwechseln mit dem ZM, welches dem BMS als Signalquelle dient. Es bildet ein Spannungsersatzschaltbild ab, welches in Abschnitt 1.5.4.2 zu sehen ist. Hauptaufgabe des VCMs ist die Berechnung des spannungsbasierten Ladezustandes `SocVb`. Zusätzlich übernimmt das VCM weitere Funktionen, wie die Berechnung der Innenwiderstände der Zelle, oder die Voraussage der Kapazitätsänderung unter Belastung. Das VCM benötigt beim Systemstart eine Reihe von Initialwerten, welche bei Bedarf aus dem NVM gelesen werden. Im Falle von unvorhergesehenen Betriebszuständen, in denen das VCM keine NVM-Werte lesen kann, wird auf interne Default-Startwerte zurückgegriffen.
Das VCM setzt sich aus folgenden Modulen zusammen:

1. Voltage-Current-Synchronisation (VltgCurrSynch)
2. Resistance Calculation (ResCalc)
3. Hysteresis Calculation (HystCalc)
4. Open Circuit Calculation (OcvCalc)
5. State of Charge – Voltage based (SocVb)

Spannungsersatzschaltbild: Die Funktion des VCM basiert auf Grundlage eines elektrischen Ersatzschaltbildes (siehe Abschnitt 1.3.3), welches aus einer Spannungsquelle (OCV), einem Widerstand (R_i) und zwei RC-Gliedern (RC_{pol_short} und

RC_{pol_long}) besteht. Die Spannungsquelle stellt die Systemspannung im lastfreien Zustand dar (I = 0), der Innenwiderstand und die RC-Glieder modellieren die Spannungsantwort nach Zellbelastung (vergleiche bspw. [SMW11, ECKF11, Zha10]). Die zwei seriellen RC-Gliedern bilden den Funktionsverlauf eines Übertragungssystems zweiter Ordnung ab, womit das reale Relaxationsverhalten der Zelle ausreichend genau modelliert werden kann (siehe Abschnitt 1.5.3.11).

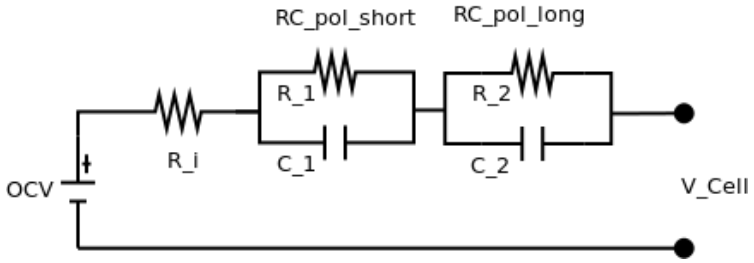

Abbildung 1.22: Das zu Grunde liegende Impedanzersatzschaltbild mit zwei RC-Gliedern

Voltage-Current-Synchronisation: Der Name dieses Moduls beschreibt nicht die tatsächliche Funktionalität, da sie im Funktionsdesign des VCM mittlerweile verändert wurde. Im vorliegenden VCM berechnet das Modul die Ladezustandsänderung pro Zyklus anhand des Eingangsstroms im Rahmen seines Toleranzbereichs. Verwendet werden die Modulsignale beim SOC, beim SOH und bei der Berechnung der OCV-Hysterese.

Hysteresis-Calculation: Der Hysteresewert (`Hyst`) stellt einen Faktor zwischen 0 und 1 dar, der nach dem Vorzeichenwechsel des Eingangsstromes, den Wechsel von der Lade- zur Entladetrajektorie (SOC/OCV) und umgekehrt ermöglicht. Ein `Hyst` von 0 würde demnach bedeuten, dass als Spannungswert die Entladetrajektorie, 1 die Ladetrajektorie gültig ist. Zwischen den Werten findet eine stetige, lineare Interpolation der beiden Trajektorien statt (siehe Abb. 1.23). Der Einfluss der Berechnung ist in Gleichung 1.4 zu sehen, wobei die Berechnung des SOCs ebenfalls davon abhängt. Ausgehend von einem Start-`Hyst` wird das Hysteresesignal aus der Systemkapazität und der aus der Systembelastung resultierenden Ladezustandsänderung berechnet. Der Startwert wird entweder aus dem NVM gelesen, bei Simulationsstart definiert oder als letzte Möglichkeit aus einem Default-Wert gelesen.

OCV-Calculation: Das Modul der OCV–Berechnung basiert auf der Abbildung jener *Look-up Tables* (LUT), die aus gegebenem SOC und Zelltemperatur einen Ruhespannungspegel der Zelle beschreiben. Dies geschieht in Lade- und Entladerichtung, in Abhängigkeit des momentanen Hysteresewerts. Bei Systemstart wird für die OCV–Initialisierung auf die im NVM gespeicherten SOC-Werte zurückgegriffen bzw. im Fehlerfall wird sie direkt aus der gemessenen Spannung ermittelt. Der SOC kann für die Berechnung von spannungs- oder strombasiertem Ursprung sein.
Die OCV-Berechnung findet für einen minimalen (`OcvMin`), maximalen (`OcvMax`) und durchschnittlichen Wert (`OcvAvg`) in folgender Weise statt (allgemeine Form):

$$OCV = (SocChg - SocDchg) * Hyst + SocDchg \tag{1.4}$$

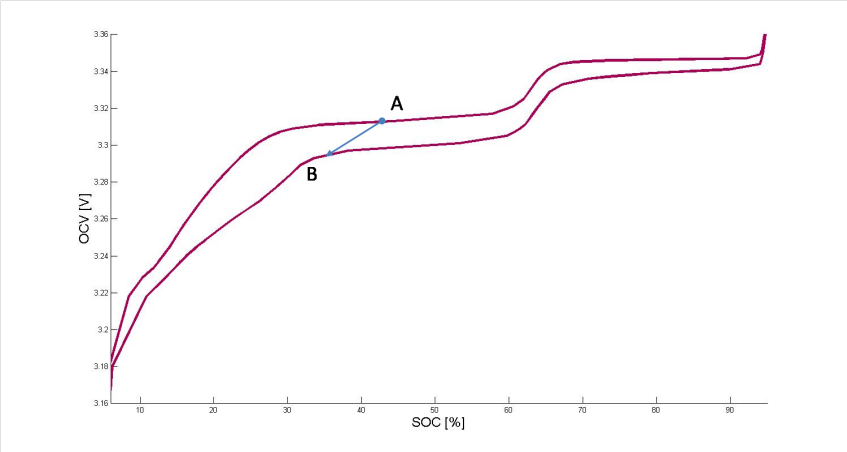

Abbildung 1.23: Die Zelle wird in diesem Beispiel bis zum Punkt A geladen, danach konstant entladen. Bis zum Erreichen von Punkt B, errechnet Gleichung 1.4 einen OCV-Wert zwischen den Trajektorien

Verwendet wird die OCV für die Berechnung des Batterieinnenwiderstands, der SOP-Stromlimits und für die SOH–Berechnung.

Resistance-Calculation: In diesem Modul werden Widerstandsvorhersagen auf Basis aktueller Messgrößen berechnet. Zum einen handelt es sich um jene Widerstände, die für den Leistungsverlust in der Zelle bei minimaler, maximaler und durchschnittlicher Zellspannung verantwortlich sind. Zum anderen um jene Widerstandswerte, die für die gegebenen Lang- und Kurzzeitvoraussagen der im SOP berechneten, möglichen Batterieströme zu erwarten sind. Grundlage des Moduls bilden die 3D-

LUTs, welche vom SOC, der Temperatur, der Zellalterung und dem Lithium-Plating (siehe Abschnitt 1.4) limitierenden Strom abhängen. Letzterer kommt aus dem SOP-Modul, dessen Einfluss in weiterer Folge aber nicht näher betrachtet wird.

Voltage-Based SOC-Calculation: Der `SocVb` wird wie der `SocCb` als Prozentwert zwischen 0 und 100 dargestellt. Ebenso die Werte der Signalfehler, die ein Maß für die Unsicherheit zum gegebenen Zeitpunkt darstellen (siehe Abschnitt 1.5.3.6).
Der `SocVb` wird anhand von vier Signalen beschrieben. Das Modul berechnet:

- einen minimalen (`SocVbMin`), maximalen (`SocVbMax`) und den durchschnittlichen (`SocVbAvg`) spannungsbasierten Ladezustand unter Berücksichtigung aller Zellen im Zellverbund

- eine periodische `SocVb`-Ausgabe einer jeden einzelnen Zelle (`SplSoc`) samt Index.

Die `SocVb`-Berechnung liegt der minimalen, maximalen und durchschnittlichen Zellspannung zugrunde, auch dann, wenn der Einfluss eines Belastungsstromes noch nicht abgeklungen ist. Deshalb wird im Modul auf einen Hilfszustand zurückgegriffen, die virtuelle OCV. Demnach werden im belasteten Modell die Spannungsabfälle an den Innenwiderständen von der Zellspannung abgezogen, um die übrige Quellspannung (OCV) zu erlangen, aus der per LUT ein Wert für den `SocVb` und `SocVbErr` erzeugt wird.
Die Abbildung des `SocVb` erfolgt mit Hilfe von 2D-LUTs aus dem virtuellen OCV und der Systemtemperatur, welche anschließend vom aktuellen Hysteresewert beeinflusst wird. Dessen Signalfehler-Werte für `Min-`, `Max-` und `SplSoc` werden in positive (steigende Signalanteile) und in negative (fallende Signalanteile) Richtung generiert (siehe Abschnitt 1.5.3.6). Die für die `SocVb` Berechnung notwendigen Spannungsabfälle am Innenwiderstand und den beiden Polarisationswiderständen, werden in Abhängigkeit von `SocCb`, Eingangsstrom und Systemtemperatur aus 3D – LUTs gelesen. Aus Gründen eingeschränkter Messgenauigkeiten (siehe Abschnitt 1.5.3.11), wird ein generierter `SocVb` erst für die Ladezustandsbestimmung relevant, wenn seine Berechnung einem kurzen, konstanten Belastungsstrom zu Grunde liegt. Konkret sollte der Strom seinen Betrag innerhalb von wenigen Sekunden nicht stark ändern. Innerhalb der Zeitbeschränkung wird deswegen der Fehlerwert des `SocVb` hohe Werte annehmen, um ihn für die Korrektur unattraktiv zu machen (siehe Korrektur in Abschnitt 1.5.4.3).

1.5.4.3 SOC-Modul

Dieses Modul berechnet den Ladezustand der gesamten Batterie, ohne die einzelnen Zellen zu berücksichtigen. Um auch nach längeren Fahrzyklen einen korrekten Wert garantieren zu können, führt das integrierte *Current Activity Monitoring* [MSB12] die SOC-Korrektur durch (siehe Korrektur in Abschnitt 1.5.4.3).

SOC-Bestimmung: Bereits bei der Inbetriebnahme des Batteriesystems muss der korrekte Ladezustand des Zellverbandes dargestellt werden können. Seiner Berechnung liegen zwei unabhängige Varianten zugrunde.
In der ersten Variante wird der Endwert des letzten Testlaufs, falls vorhanden, aus dem NVM gelesen. Voraussetzung dafür ist, dass am Ende des letzten Fahrzyklus ein SOC ermittelt und gespeichert werden konnte. Sollte aus Fehlergründen der NVM-Wert nicht zur Verfügung stehen, wird der Start-SOC-Wert vom VCM aus der gemessenen Spannung berechnet, unter der Annahme, dass diese den OCV repräsentiert.
Während des Batteriebetriebes wird der SOC entweder aus Stromintegration (siehe Abschnitt 1.5.3.5) oder aus den OCV-Werten ermittelt (siehe Korrektur in Abschnitt 1.5.4.3). Die Stromintegration stellt im Betrieb die primäre, echtzeitfähige Methode zur Bestimmung des SOCs dar. Die spannungsbasierte SOC-Bestimmung erfolgt mit Hilfe eines komplexen Algorithmus, der mehrere Kriterien berücksichtigt, um einen gültigen SOC abbilden zu können (siehe Abschnitt 1.5.4.2).
Als Output generiert dieses Modul je einen strombasierten Ladezustand (`SocCb`) für die Zelle mit dem niedrigsten (`SocCbMin`) und dem höchsten (`SocCbMax`) Wert im Zellverbund (jeder besitzt zusätzlich Signalfehler-Werte). Ebenso den Durchschnittswert über alle Zellen (`SocCbAvg`). Der Batterie-SOC, welcher den Ladezustand der Batterie letzten Endes präsentiert, wird aus SOC-Min, -Max und –Avg gebildet. Den hohen SOC-Bereich bestimmt der SOC-Max, den unteren der SOC-Min. Im mittleren SOC-Bereich wird der SOC-Avg für den Batterie-SOC verwendet.

SOC Korrektur: Die Entscheidung, welche Methode zur SOC-Berechnung angewandt wird, fällt wie bereits erwähnt das Current Activity Monitoring [MSB12]. Bei Systemstart wird das Monitoring erstmals aktiv und überprüft die mögliche SOC-Initialisierung, also ob die NVM-Werte verwendet werden können, oder sie von VCM angefordert werden müssen. Fehlende NVM-Werte sollten aber im Betrieb einen seltenen Fehlerfall bilden und werden deshalb nicht weiter behandelt.

Im Falle von gültigen Eingangssignalen von Strom, Spannung und Temperatur beginnt der Algorithmus, den SOC aus der Stromintegration (`SocCb`) in Echtzeit zu berechnen. Anderenfalls werden die bereits bekannten SOC-Werte eingefroren und nur deren Signalfehler inkrementiert (siehe Abb. 1.25). Während des Fahrbetriebes wird in Abhängigkeit mehrerer Kriterien überprüft, ob eine Korrektur auf den spannungsbasierten `SocVb` möglich ist. Dafür startet der Algorithmus den Sanity-Check mit integriertem Pausibility-Check (siehe Abb. 1.25).

Sanity-Check: Dieser Algorithmus prüft die Möglichkeit einer SOC-Korrektur. Wichtig dafür sind:

- ein klar definierter, konstanter Strompuls (siehe Abb. 1.24) bzw. eine ausreichend lange Ruhezeit nach Zellbelastung

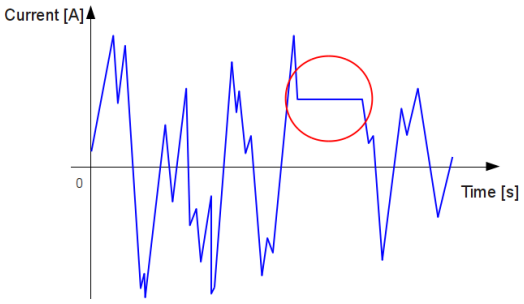

Abbildung 1.24: Darstellung einer konstanten Zellbelastung während eines Fahrzyklus

- eine Mindestdauer seit der letzten erfolgreichen Korrektur
- ein minimaler SOC-Korrektur-Schritt mit Rücksicht auf die korrigierte SOC-Richtung.

Pausibility-Check: Als Teil des Sanity-Checks wird überprüft:

- ob die berechneten Werte für Max, Avg und Min in fallender Reihenfolge vorliegen
- ob keine Überschneidungen der SOC-Toleranzbereiche vorliegen
- ob die `SocVb`-Errors nicht auf ihren Extremwerten (0 oder 100) liegen
- der `SocVb` in seinem Gültigkeitsbereich zwischen 10% und 90% liegt
- die Differenz von `SocVb` und `SocCb` dasselbe Vorzeichen wie der Eingangsstrom hat.

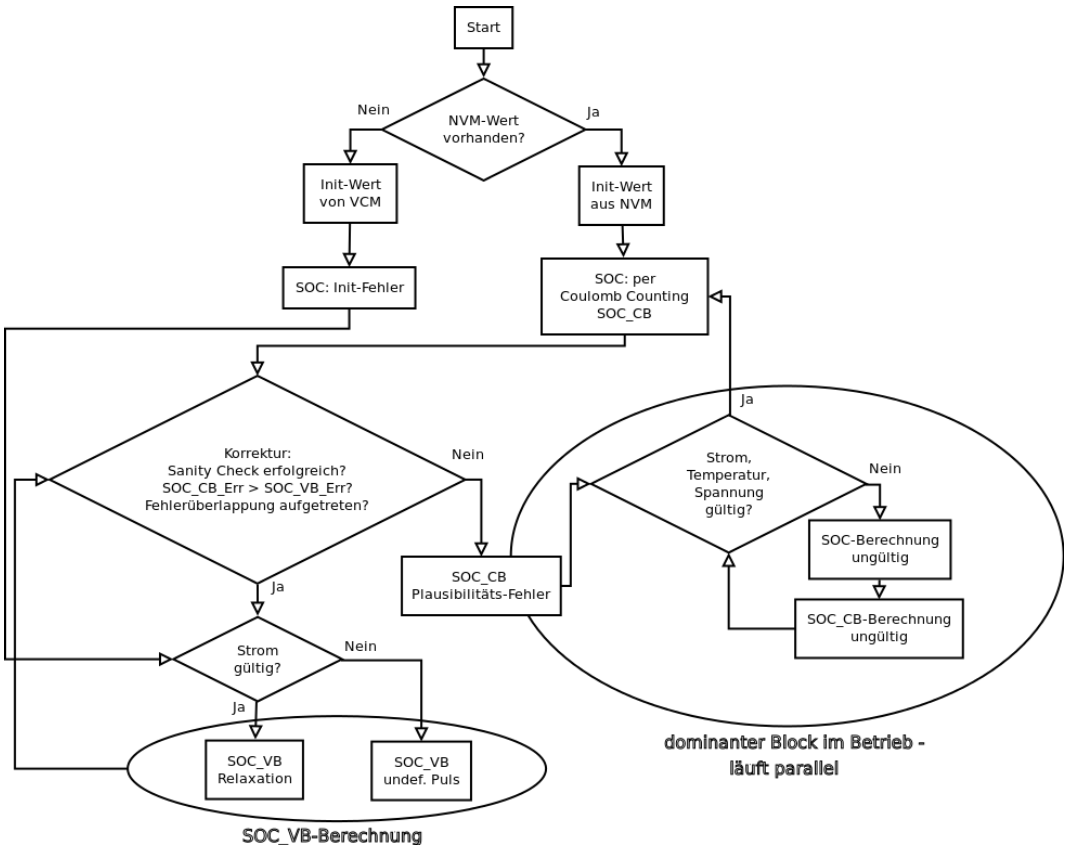

Abbildung 1.25: Flussdiagramm des SOC-Korrekturalgorithmus

SOC-Toleranzbereiche werden überschnitten, wenn der `SocCb` abzüglich seines Fehlers größer als der `SocVb` zuzüglich seines Fehlers ist und umgekehrt. Das gilt für die Min- und Max-Werte dieser Signale. Beim längeren Laden an der Ladestation wird der SOC anhand der Dauer und der Amplitude des Konstantstroms bestimmt.

1.5.4.4 SOP-Modul

Im SOP-Modul werden die Limitierungen für Strom und Spannung berechnet, welche für die Betriebssicherheit der Batterie notwendig sind (siehe Abschnitt 1.5.3.2). Notwendig dafür sind Widerstands- und OCV-Informationen vom VCM, Eingangsstrom, -spannung und –temperatur und eine Reihe von Statussignalen zur Kommunikation zwischen den Modulen. Limitiert werden die Höhe und die Pulsdauer des Eingangsstroms, die Eingangsspannung, die vom System geforderte Leistung und die Lithium-Plating (siehe Abschnitt 1.4) verursachenden Ströme über die Tempe-

raturbereiche. Das Modul berechnet die Limitierungsgrenzen unter Berücksichtigung der überschätzten, bautechnisch bedingten, Leistungsfähigkeit. Das entspricht einer künstlichen Anhebung der berechneten Signallimitierungen. Damit wird erreicht, dass unter realen Fahrzyklen die tolerierbaren Leistungsbereiche besser ausgenutzt werden können.

Die folgenden Module befinden sich im verwendeten B-Muster-BMS noch in der Entwicklung, werden im Rahmen dieser Arbeit nicht näher betrachtet und daher zur Vollständigkeit nur kurz beschrieben.

1.5.4.5 SOH-Modul

Das Modul der Zellalterungssimulation soll in Zukunft einen virtuellen Stundenzähler modellieren, welcher auf die Batterieparameter Einfluss nehmen soll. Die Berechnung basiert auf einer Kombination von kalendarischer und zyklischer Alterung mit den Informationen des Energiedurchsatzes der im Fahrzeug integrierten Ladeeinheit.

1.5.4.6 CapaDet-Modul

Die in (siehe Abschnitt 1.5.3.9) beschriebene Funktion wird von der SOH- und der Balancing-Berechnung benötigt. Die Kapazitäten werden vom Algorithmus zum geeigneten Zeitpunkt für jede Zelle bestimmt.

1.5.4.7 CellBal-Modul

Für die in (siehe Abschnitt 1.5.3.8) beschriebene Balancing-Funktion muss das Modul von jeder Zelle im System den Ladezustand und die verfügbare Kapazität kennen.

1.6 Stand der Technik

Das folgende Kapitel gibt einen kurzen Einblick in aktuelle Forschungsarbeiten und deren Erkenntnisse für ähnliche Aufgabenstellungen wie jene dieser Arbeit, die in Abschnitt 1.7 beschrieben wird. Vorgestellt werden Varianten der Zellmodellierung, Verfahren zur Zell-Parametrierung und Versuche mit alternativen Zell-Technologien.

Hauptaugenmerk wird dabei auf den Stand der Technik im Bereich der Lithium-Ionen Technologie gelegt, vor allem für Hochstromanwendungen. Dennoch wurden auch wertvolle Erkenntnisse mit anderen Zellchemien im Rahmen der Batteriesimulation erlangt, die in diesem Abschnitt erwähnt werden. Auf die Abhängigkeit der Ergebnisse von der jeweiligen Chemie muss aber geachtet werden.
Um das elektrochemische Verhalten von galvanischen Zellen nachvollziehen und vorhersagen zu können, wurden bereits im Jahr 1965 mathematische Zusammenhänge veröffentlicht. Sie bilden die Eigenschaften des Lade- und Entladeverhaltens in Abhängigkeit von Innenwiderständen und Polarisationseffekten ab, um die Herstellung optimieren und letztendlich Gewicht und Größe der Zelle reduzieren zu können [She63]. Die damals verwendeten mathematischen Methoden dienen noch heute als Grundlage für die Modellierung von komplexen Batteriesystemen.

1.6.1 Beispiele der Modellbildung

Im Vergleich zur vorliegenden Arbeit, bietet [HLS11] einen Einblick in einen aktuellen, praktischen Prozess, eine Lithium-Polymer-Akkumulator, bestehend aus sieben $40Ah$ Zellen, zu modellieren. Anhand eines einfachen Modells wird die Simulation von SOC, OCV und der Systemspannung unter Belastung dargestellt, ohne Rücksicht auf Veränderungsfaktoren wie z.B. den Temperatureinfluss oder die Zellalterung.

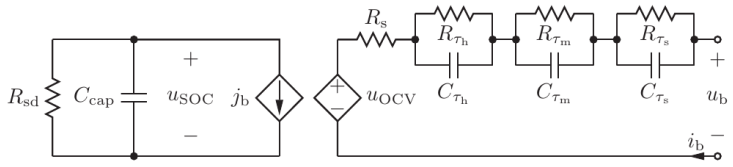

Abbildung 1.26: Impedanzersatzschaltbild mit drei RC-Gliedern nach [HLS11]

Die Kapazität wird im Modell anhand eines Kondensators realisiert, parallel dazu ein ohmscher Widerstand für die Nachbildung des Selbstentladungseffektes (siehe Abb. 1.26). Den Innenwiderstand der Batterie bildet ein einzelner ohmscher Widerstand ab, welcher in Verbindung mit drei RC-Gliedern das Spannungsverhalten unter Belastung beschreibt. In diesem Paper stellt die Wahl von drei RC-Gliedern einen guten Kompromiss zwischen Simulationsgenauigkeit und Komplexität dar, soll aber in [Zha10] genauer evaluiert werden. Zusätzlich wird die Möglichkeit beschrieben, die Spannungs- und Widerstandsabhängigkeit über Polynomfunktionen höhe-

rer Ordnung mittels Fitting (siehe Abschnitt 1.5.3.11) nachzubilden. Im Experiment wird nur das nachfolgend beschriebene Messverfahren zur Gewinnung der Parameter angewendet.
Die Idee des Verfahrens ist mit jenem aus Abschnitt 1.5.3.10 vergleichbar, wobei eine vollständig geladene Zelle schrittweise Entladen wird und nach jedem Schritt eine Ruhepause der Zellbelastung einzuhalten ist. Im Experiment wurde eine Pause von 30 min als ausreichend lang befunden, um akzeptabel genaue Messwerte aufnehmen zu können (siehe Abschnitt 1.4). Der zulässige Spannungs-Betriebsbereich der Zelle wird in 10% - SOC-Schritten in Lade- und Entladerichtung vermessen und in LUTs in Zusammenhang gebracht. Zusätzlich wurde ein Unterschied im Zellspannungsverhalten zwischen den Laderichtungen detektiert (vergleiche Hysterese in Abschnitt 1.5.4.2).
Zur Validierung der Modellfunktion wird das Leistungsprofil eines Lastkraftfahrzeugs im Bergbau gewählt, weil ständig ändernde Steigungen im Fahrzyklus für die Messung von Vorteil sind. In Abhängigkeit der Systemspannung wird daraus in Echtzeit das Stromprofil erstellt, welches gemeinsam mit der gemessenen Systemtemperatur das Modell bedatet. Im Vergleich zur gemessenen Systemspannung bietet diese Modellvariante eine akzeptable Nachbildung mit Schwächen in den SOC-Randbereichen, verursacht durch zu wenige Messpunkte im steilen Funktionsverlauf.
Für die akzeptable Modellierung von Hysterese- und Relaxationseffekt haben sich elektrische Ersatzschaltbilder von Impedanzmodellen (vergleiche die Abschnitte 1.3.3 und 1.5.4.2) durchgesetzt, die für verschiedene Batterietypen frei parametrierbar sind [Zha10]. In der Literatur sind sie sehr häufig als Thevenin-Theorem zu finden.
Ein wichtiger Aspekt neben der Modellierung des Innenwiderstands der Batterie, ist die Nachbildung des Relaxationsverhaltens mittels RC-Glieder, wie zuvor aus [HLS11] beschrieben. Durch die Erhöhung der RC-Glieder-Anzahl steigt die Rechengenauigkeit der Simulation, die Spannungsantwort an jene des Realsystems anzugleichen. Jedoch erhöhen sich ebenso der Berechnungsaufwand und die Modellkomplexität um ein Vielfaches.
[Zha10] evaluiert diesen Zusammenhang, um eine Abschätzung für den besten Kompromiss in einer Zellsimulation zu finden. Per Konstantstrom-/Konstantspannungspulsen wird die Zellcharakteristik gemessen, woraus über Fitting-Algorithmen (siehe Abschnitt 1.5.3.11) die Parameter der RC-Glieder gefunden werden. Dem folgt eine Fehleranalyse, welche die Unterschiede von gemessener zu simulierter Spannung gegenüberstellt und den Durchschnitts- bzw. Worst-Case Fehler darstellt. Im Versuch wird dieses Verfahren von Modellen mit einem bis fünf RC-Glieder durchgeführt. Aus dem Versuch wird ersichtlich, dass beim Wechsel von einem auf zwei RC-Glieder der höchste Genauigkeitsgewinn erzielt wird, der bei einer weiteren Erhöhung nur mehr

gering ansteigt.
In [Zha10] wird eine Berechnungsaufwandsabschätzung für zwei Ergebnisdatentypen unterschiedlicher Genauigkeitsauflösung für die fünf Modellvariationen durchgeführt. Bei der Berechnung der optimalen RC-Anzahl wird zum einen die Gewichtung zwischen Genauigkeit und Echtzeitfähigkeit, zum anderen die akzeptablen Grenzen von Berechnungsfehlern und Berechnungszeiten berücksichtigt. Unter der Voraussetzung, dass das Modell keine Fehler kleiner 10% generiert und nicht länger als 2s rechnet, wird für eine eher ausgeglichene Gewichtung der beiden Kriterien ein RC-Optimum in der Nähe der zwei RC-Glieder gefunden (siehe Abb. 1.27).

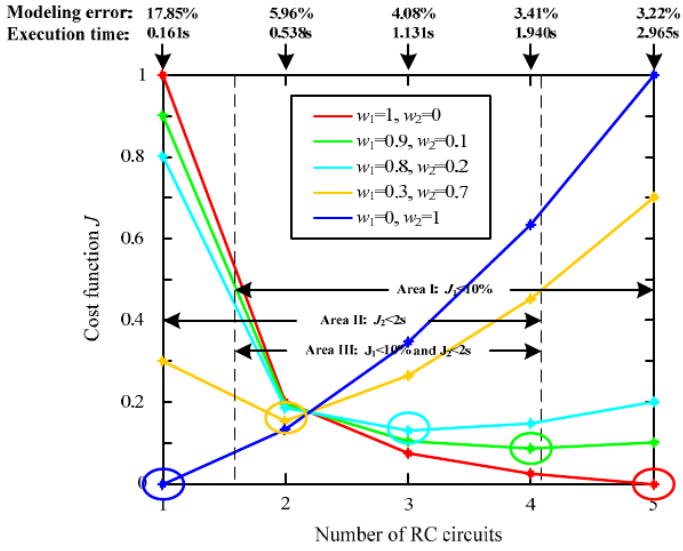

Fig. 9. Cost function value with different w_1 and w_2 settings.

Abbildung 1.27: Ein Auszug der Evaluierung aus [Zha10]. Die Rechengenauigkeit wird über die Anzahl der RC-Glieder aufgetragen

1.6.2 Erläuterung verschiedener Verfahren zur Modell-Parametrierung

Im Rahmen einer Gegenüberstellung von zwei unterschiedlichen Varianten der Parametrierung aus [ECKF11], soll die maximal erreichbare Rechenperformance mit der höchsten Vereinfachung des Modells evaluiert werden. Dafür wird die Komplexität des Modells während des Experiments von keinem bis zwei RC-Glieder schrittweise erhöht. Der steigende Aufwand der Parametrierung bringt, wie in [Zha10] erwähnt, eine erhöhte Simulationsgenauigkeit in den Bereichen Simulationsspannung und SOC mit sich. Der SOC wird im Experiment aus Spannungsmessungen am Innenwiderstand und den optionalen RC-Gliedern, bestehend aus den Polarisationswiderstän-

den und Kapazitäten berechnet, deren Werte die Modellparameter darstellen.
Die Parameter werden entweder durch nachfolgend beschriebene Messverfahren an 23 Punkten in LUTs abgebildet, oder anhand eines *linear Models* (LIM) dargestellt. Im ersten Fall wird das System in der Simulation mit 23 Strompulsen in der Höhe von je $20A$ entladen und die Spannungsantwort an den Widerständen und den Kapazitäten gemessen (OCV). Im zweiten Fall wird, von SOC=100 weg, über einen kompletten Entladevorgang, ein linearer Zusammenhang zwischen SOC und den Bauteilen generiert.
Der Unterschied im Aufwand der Parametrierung wird sofort ersichtlich, wo 23 Schritte einem Einzigen gegenüberstehen. Für die Evaluierung der Simulationsgenauigkeit wird der erwähnte $20A$ Entladepuls-Zyklus und der amerikanische FTP72 Fahrzyklus über $12.07km$ [Ha12, Wik13] verwendet. Im Vergleich zum gemessenen Spannungsverlauf wurde der quadratische Fehler der Simulation über die Zyklendauer errechnet.
Das Experiment kommt zu dem Ergebnis, dass unter Vernachlässigung verschiedenster Temperatureinflüsse (siehe Abschnitt 1.4) in der Simulation, jenes Model mit einem RC-Glied und der linearisierten Innenwiderstandskennlinie über den SOC-Bereich den besten Kompromiss liefert. Der im Vergleich geringe Aufwand der Parametrierung kann dabei eine akzeptabel genaue Simulation eines realen Fahrzyklus erreicht werden.
[ECKM11] setzt diese Arbeit fort, ergänzt sie mit dem Verfahren einer vereinfachten Parametrisierung im Falle der Kombination des LIM mit einem einzigen RC-Glied. Dies wird erreicht, indem das Modell mit längeren Strompulsen entladen wird, um das volle Ausmaß der Spannungsantwort der Zelle messen zu können. Der Innenwiderstand errechnet sich somit aus dem ohmschen Gesetz, die Werte des RC-Glieds aus den daraus folgenden Zusammenhängen.

1.6.3 Alternative Methoden zur Gewinnung von Parametern diverser Batteriemodelle

[CH08] berichtet über eine alternative Berechnungsmethode für die Parameter Kapazität, SOC und SOH, für *Valve Regulated Lead Acid* (VRLA) Zellen[3], aber darauf aufbauend auch für Li-Ion Zellen. Die Idee hinter der Berechnung basiert auf einer 2-Puls Test-Methode während der Zellentladung.
Der erste Strompuls dient dazu, die Charakterisierung von der im praktischen Fall unbekannten Belastungsvergangenheit unabhängig zu machen, wie im Artikel demonstriert wurde. Nach einer kurzen Relaxationszeit und der abschließenden Mes-

[3] Bleiakkumulator

sung des OCVs zu diesem Zeitpunkt wird die Batterie mit demselben Puls nochmals belastet. Aus der Strompulshöhe (I), der gemessenen Spannungsantwort des ersten Pulset nach $10s$ (V_{Max}) und dem Spannungsabfall des zweiten Pulses (ΔV_2) können Ladezustand, Kapazität und das Batteriealter bestimmt werden (siehe Abb. 1.28).

Die Ladezustandsberechnung liegt einem linearen Zusammenhang von OCV und

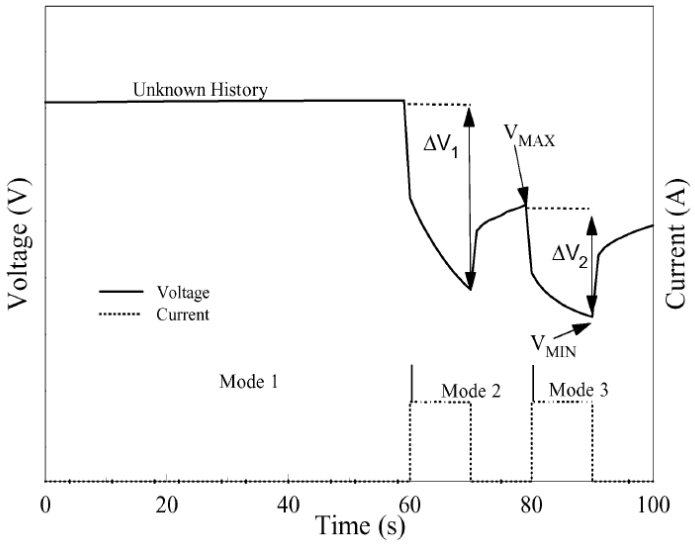

Abbildung 1.28: Darstellung der 2-Puls Test-Methode aus [CH08]. Der Spannungsverlauf wird anhand von zwei Entladestrompulsen gebildet

SOC aus dem Datenblatt des Herstellers zugrunde, ein Vorteil von Bleiakkumulatoren. Der SOC wird somit aus V_{Max}, einem Spannungsoffset und den Steigungsparametern berechnet:

$$SOC = \frac{V_{Max} + \beta - OCV}{\alpha} \tag{1.5}$$

Dies stellt eine einfachere Methode als jene mittels Coulomb Counting dar (siehe Abschnitt 1.5.3.5), da die Rekalibrierung des Stromsensors ausfällt und keine zeitliche Integration über den Stromverlauf erfolgt. Die Kapazität erhält man durch Messung des Spannungsabfalls ΔV_2 als Reaktion nach unterschiedlichen Strompulsen. Im Experiment werden acht Testzellen unterschiedlicher Kapazität mit vier aufsteigend hohen Pulsen belastet. Im Anschluss wird mit den gewonnenen Informationen und zwei zusätzlichen Parametern, die mittels Methode der kleinsten Quadrate aus dem Werteverlauf berechnet wurden, die C-Rate bestimmt. Der Zusammenhang von ΔV_2 und der C-Rate wird wieder als linear angenommen. Daraus erhält man in Abhängigkeit des Strompulses die Kapazität und in weiterer Folge auch das Alter der Batterie [CH08]. Am Ende des Papers wird der große Nachteil der 2-Puls Methode,

vor allem bei der Anwendung von Li-Ion Zellen, erläutert. Er entsteht durch den notwendigen linearen Zusammenhang von SOC zu OCV. Da dieser nur bestimmten OCV-Regionen existiert, kann die Methoden auch nur in diesen Bereichen erfolgreich angewendet werden, dort aber mit dem großen Vorteil der fast parameterlosen, echtzeitfähigen Zustandsbestimmung.
[YGB08] berichtet von der Problemstellung, dass HEVs nicht in ihrem kompletten SOC-Bereich betrieben werden, trotzdem eine steigende Anzahl von Kontrollparametern, länger andauernde Kalibrierungszeit und steigenden Kosten in der Entwicklung vorweisen. Deswegen wird nach einem reproduzierbaren Parametrier-Algorithmus gesucht, der Charakteristiken identifiziert und davon Prognosen bzw. Diagnosen unter akzeptablem Aufwand liefern kann. Im Vordergrund stehen gängige, echtzeitfähige SOC-Berechnungsmethoden, deren SOC-Messabweichungen 2% nicht überschreiten.
Als weiteren Ansatz zur effektiven Batteriemodell-Parametrierung versucht [YGB08], mittels genetischem Algorithmus den SOC/OCV-Zusammenhang abzubilden. Er sucht die Parameter eines vorgegebenen Modells um dessen Ergebnisfunktionen an ausgewählte, gemessene Datensätze anzugleichen.
Der Algorithmus sucht aus einer Menge möglicher Lösungen jenen Wert aus, der den Initialwert der Problemstellung am besten ergänzt. Eine Fitnessfunktion, welche an die Problemstellung angepasst ist, bewertet die möglichen Lösungen in Richtung Ergebnisplausibilität mit einer höheren Fitness, um zusätzlich ihre Auswahlwahrscheinlichkeit zu erhöhen. Im Anschluss vereint der Algorithmus, quasi per Zufall, zwei ausgewählte, mögliche Lösungen, deren Auswahlwahrscheinlichkeit aufgrund der höheren Fitness angehoben wurde, um zwei mögliche Lösungen der nächsten Generation zu schaffen (Mutation). Diese Ergebnisevolution setzt sich fort, bis der Algorithmus ein Abbruchkriterium erreicht und die fitteste, mögliche Lösung als Ergebnis markiert.
In [YGB08] wird der Algorithmus zur Parameteridentifikation angewandt, um die nichtlineare SOC/OCV-Trajektorie im Lade- und Entladefall einer NiMh-Zelle nachzubilden. Die Zelle wird auf Grundlage eines Impedanz-Ersatzschaltbildes mit einem RC-Glied modelliert, dessen Parameter vom genetischen Algorithmus zu identifizieren sind. Dabei wird der modellierte Spannungsverlauf des Zellmodells mit gemessenen Zellspannungen an ausgewählten Punkten verglichen und die Differenz durch den Algorithmus minimiert. Zwischen zwei optimierten Stützpunkten wird der Zusammenhang linearisiert. Im Experiment kommen mehrere parallelisierte Rechner zum Einsatz, um den hohen Zeitaufwand der Berechnung zu minimieren. Zur zusätzlichen Aufwandsreduktion dieses Verfahrens werden verschiedene Einschränkungen definiert, beispielsweise ein begrenzter SOC-Bereich. Die Qualität vom Ergebnis

hängt trotzdem stark von Anzahl und Dynamik der zu Grunde liegenden Datensätze ab. Das beschriebene Experiment liefert ein auf die Einschränkungen bezogenes, effizientes Verfahren zur Modellierung und Parametrierung einer HEV-Batterie, dessen Simulationsergebnisse im Gegensatz zum Coulomb Counting, im Rahmen von akzeptablen Fehlergrenzen liegen.

1.6.4 Alternative Methoden zur Berechnung des SOCs

In [ZC10] wird eine alternative, echtzeitfähige Berechnung des SOCs beschrieben, die zur Berücksichtigung der vergangenen Zellbelastung einen dafür entwickelten Algorithmus verwendet, um somit den zusätzlichen Aufwand zur Charakterisierung des Spannungsverhaltens einer Zelle zu vermeiden. Grund für den Berechnungsaufwand ist der unterschiedliche OCV-Verlauf im Lade-/Entladefall über den gesamten Ladezustandsbereich (Erklärung des chemischen Effekts in Abschnitt 1.4).
Die SOC/OCV-Abhängigkeit wird in den wissenschaftlichen Arbeiten [ZC10], [ECKF11] oder [ECKM11] mit Hilfe eines Entlade-Experiments nachgewiesen. Eine vollständig geladene Li-Ion Zelle (vom Zell-Hersteller A123 mit ca. $2.5Ah$) wird mittels $0.5A$ Pulsen entladen (zw. 2 und $3.6V$). Nach jedem Puls wird ca. eine Stunde abgewartet, um die Relaxationseffekte abklingen zu lassen und die Spannung an der Zelle messen zu können. Dasselbe Experiment muss im selben Verfahren mit Ladepulsen durchgeführt werden (vergleiche Abschnitt 1.5.3.10).
Der Algorithmus der SOC-Bestimmung trennt die SOC/OCV Trajektorie in nichtlineare und lineare Verläufe. Der lineare Verlauf befindet sich in der Charakteristik des Experiments im mittleren SOC/OCV-Bereich und wird für Lade- und Entladerichtung getrennt betrachtet. Unterschieden wird der komplette Ladevorgang (Major Loop) und der partielle Ladevorgang (Minor Loop) (siehe Abb. 1.29).
Bei der Beschreibung der nichtlinearen Verläufe am Beginn und am Ende der Trajektorie wird für Lade- und Entladerichtung je auf zeitinvariante Systeme erster Ordnung zurückgegriffen. Der komplette Major-Loop wird anschließend aus den linearen und nichtlinearen Funktionsverläufen zusammengestellt, deren Parameter aus linearem und nichtlinearem Fitting erhalten wurden. Unter der Annahme, dass der Minor-Loop dieselbe Dynamik wie der Major-Loop aufweist, wird der nichtlineare Funktionsverlauf des Major-Loops für die Startzustände des Minor-Loops übernommen. Der Algorithmus dient den angekündigten, zukünftigen Forschungsarbeiten als ausreichend genaue Modellierung des Hysterese-Effekts (siehe Abschnitt 1.4).
[CLZH07] beschreibt eine alternative Berechnungsweise des SOCs unter Berücksichtigung der realen Einflüsse wie Temperatur, Alter und die stromabhängige Innenwiderstandsänderung von Li-Ion Batterien. Dieses Verfahren soll, ohne auf langwierige

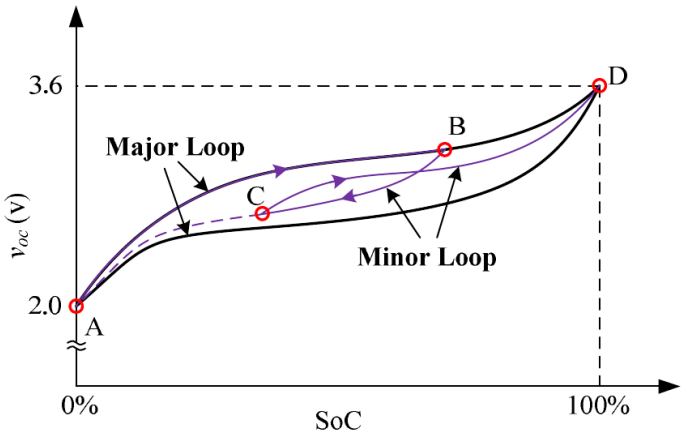

Abbildung 1.29: Major und Minor Loop nach [ZC10]

Parameterbestimmungen angewiesen zu sein, die mit der Zeit steigende Ungenauigkeit des Coulomb Counting ersetzen können. Die Berechnungen basieren auf der Grundlage, den SOC und Widerstandsverlauf während eines Lade- oder Entladevorganges in lineare und nichtlineare Bereiche des Funktionsverlaufes zu unterteilen. Die für die beiden Gleichungen notwendigen Koeffizienten werden aus Experimenten ermittelt und beschreiben das C-Raten abhängige Spannungsverhalten der Batterie. Während der Experimente wird der SOC per Coulomb Counting ermittelt, aus dem gemessenen Strom die C-Rate und aus den gemessenen Impedanzen und Batteriespannungen die Koeffizienten berechnet. Im Vergleich zu den Herstellerdaten wird bei dieser Methode festgestellt, dass für den unteren SOC- Bereich eine nichtlineare Gleichung, für den oberen Bereich eine lineare Gleichung genauere Ergebnisse errechnet. An einer Li-Ion Batterie bringt dieses Experiment die wichtige Erkenntnis, dass die Zellimpedanz im unteren SOC-Bereich rapide ansteigt, aber im höheren Bereich kaum Veränderungen zeigt. Gemessen wird sie anhand der Spannungsantworten, wenn die Zelle mittels niedrigem Wechselstrom im kHz-Bereich, in Abhängigkeit des SOCs belastet wird. Vorteil dieser Berechnungsmethode ist laut [CLZH07] die Unabhängigkeit von langwierigen Zellcharakterisierungstests (siehe Abschnitt 1.5.3.10) durch eine schnelle Bestimmung der Koeffizienten mit Hilfe von mehreren Experimenten. Nachteil des Verfahrens ist der in [CLZH07] noch nicht enthaltene Einfluss von Temperatur und Zellalterung. Dennoch wird darin viel Potential gesehen, es in Verbindung mit Coulomb Counting als zeitunabhängige SOC-Berechnungsmethode in komplexen Systemen zu verwenden.

1.7 Motivation

Das BMS bietet in seiner modellbasierten Version die Möglichkeit, die Software, die letztendlich für den Betrieb am realen Batteriesystem daraus generiert wird, vorab auf seine Funktionen zu testen. Das vorab Testen stellt einen wichtigen Schritt in der Abteilung Zellmodellierung und Funktionsentwicklung dar, da weitere Entwicklungsschritte die korrekte Funktion diverser Module voraussetzen. Aus diesem Grund wurde in einem vorangegangenen Projekt ein B-Muster BMS in Form eines MATLAB/Simulink®-Modells vorbereitet, um es mit einem zur Verfügung gestellten Zellmodell im selben Simulationsrahmen zu betreiben. Dabei generiert das ZM die Quellsignale, die das BMS bedaten. Dieser Funktionsumfang muss aber für die weitere, entwicklungsbegleitende Funktionsanalyse erweitert werden. Die Erweiterung betrifft vor allem die Generierung der Steuerungssoftware für Zellprototypen, welche die Funktionalität der Zelle, dem momentanen Entwicklungsstand entsprechend, bereitstellen soll. Solche Zellprototypen besitzen noch kein bekanntes Betriebsverhalten, welches zuvor auf Prüfständen ermittelt (siehe Abschnitt 1.5.3.10) und mit Hilfe von Parameter ins Modell eingebunden werden muss. Für den kompletten Funktionsumfang artet diese Zellcharakterisierung in ein sehr zeitaufwändiges und deshalb auch kostspieliges Verfahren aus, da bspw. die Bestimmung der OCV-Charakteristik über alle Temperaturbereiche (siehe Abschnitt 1.5.3.10) erfolgen muss. In der Regel müssen die Zellprototypen, aufgrund von knappen Zeitplänen, während der Entwicklung, innerhalb von kurzer Dauer in ein Batteriesystem integriert werden. Darin sollen sie möglichst kurzfristig einen geringen Funktionsumfang bereitstellen, um ihren momentanen Entwicklungsstand validieren zu können. Demnach ist es nicht sinnvoll, diese Zellprototypen das gesamte Charakterisierungsverfahren absolvieren zu lassen, wenn der Einsatzbereich durch eingeschränkte Betriebsbedingungen begrenzt wird. Solche Arbeitsaufwände wären demnach nicht nötig und sollten nach Möglichkeit reduziert werden können.
Kann nicht auf reale Zellprototypen zurückgegriffen werden, muss ein bereits vorhandener Parametersatz aus einer validierten Batteriesoftware in seinem Funktionsumfang an jenen des Prototypen angepasst werden, dass man einen Eindruck über die eingeschränkte Funktionalität erhält. Die Betriebsszenarien sollten dafür den Einschränkungen, die für den Betrieb des Zellprototyps getroffen wurden, entsprechen.
Bei der möglichen Überschreitung des definierten Einsatzbereichs im Testbetrieb ist ein schwer vorhersagbares Verhalten des gesamten Batteriesystems zu erwarten. Auf jeden Fall ist mit Genauigkeitsverlust bei der Berechnung diverser Zustandsgrößen wie z.B. des aktuellen Ladezustands oder der einzuhaltenden Stromlimits zu rech-

nen. Wie groß dieser Genauigkeitsverlust tatsächlich im Vergleich zum vollständig charakterisierten System ist, muss zum Vergleich dargestellt werden können.

1.7.1 Aufgabenstellung

Die aus dem vorangegangenen Seminarprojekt „Einbindung eines PHEV-Serienbatteriesteuerungsmodells in eine Funktionsentwicklungsumgebung“ entstandene Arbeit soll im Funktionsumfang derart erweitert werden, dass die Testergebnisse vom realen Batteriesystem am Prüfstand in einer Simulationsumgebung nachgestellt werden können. Dabei bleibt die Voraussetzung bestehen, dass diese Simulationsumgebung neben MATLAB/Simulink® keine weiteren lizensierte Entwicklertools, wie z.B. die Hardwareimplementierungssprache TargetLink®, benötigen soll, um auf Standard-Entwicklungsrechnern (siehe Abschnitt 2.1.1) der MAGNA Steyr Fuel and Battery Systems betrieben werden zu können. Die am Prüfstand aufgezeichneten Tests und Fahrzyklen, die der Simulationsumgebung als Signalquelle dienen sollen, müssen zu diesem Zweck einen Anpassungsprozess durchlaufen. Für die Simulationsumgebung selbst müssen ebenso weitere Funktionalitäten entworfen werden, um eine annähernd genaue Simulation durchführen zu können. Aufgrund des frühen BMS-Entwicklungsstadiums (B-Muster), kann es vorkommen, dass im Laufe der Projektarbeit eine unvorhergesehene Fehlfunktion im Modell auftritt, die den weiteren Projektfortschritt negativ beeinflussen kann. Entwicklungsbegleitend muss deshalb auf die korrekte Funktion des BMS-Modells geachtet und im Fehlerfall darauf reagiert werden.
Eine zusätzliche Funktionalität wäre die Anpassung der Softwareparameter an eingeschränkte Betriebsbereiche von Zellprototypen. Da der Prozess der Zellcharakterisierung in erster Linie die Parameter des VCMs ermittelt, soll die Reduzierung des Betriebsumfangs des BMS auf dieses Modul begrenzt werden, z.B. für die Eingangsgrößen Temperatur, Lade-/Entladestrom oder Zellspannung. In diesem Fall wird, mittels Manipulation des Parametersatzes, ein direkter Bezug auf die Bedatung der Simulink®-Bauteile im VCM hergestellt. Deshalb kann auch nicht mehr auf die Geschwindigkeitsoptimierung der Simulation aus dem vorangegangenen Projekt Rücksicht genommen werden. Eine Vereinfachung der Simulationsumgebung ist deswegen in Betracht zu ziehen. Trotzdem soll ein vereinfachtes VCM-Modell die vollständige Eingangssignal-Bedatung wie im kompletten System erhalten, um plausible Ausgangssignale generieren zu können und eine Analyse einer optional reduzierten Systemfunktionalität zu ermöglichen.
Im Rahmen der Analyse des Verhaltens einer fiktiven Prototypensoftware mit eingeschränktem Parametersatz soll deswegen eine Möglichkeit geschaffen werden, die ge-

nerierten Ausgangssignale der Simulation mit Referenzsignalen zu vergleichen. Diese Referenzsignale können gemessene Fahrzyklen vom Prüfstand, oder spezielle Tests zur Überprüfung der verbleibenden Funktionalität sein. Die Tests sollen so gewählt werden, dass die Auswirkung des reduzierten Parametersatzes auf nicht definierte Betriebsbereiche in der Ergebnisdarstellung sichtbar wird.
Der generierte Parametersatz soll in einer Form entstehen, aus der mit wenig Aufwand eine Software generiert werden kann, die auf realer Hardware lauffähig ist, ggf. mit eingeschränktem Betriebsumfang.

1.7.2 Ziele

Zur Übersicht werden die Ziele dieses Projekts nochmal zusammengefasst:
Die Simulink®-Version des B-Muster-BMS (siehe Abschnitt 1.3.6), die keine Abhängigkeiten von TargetLink® beinhaltet, soll in einer generierten Simulink®-Simulationsumgebung von einem Zellmodell, nach Vorbild des realen Systems am Prüfstand, bedatet werden. In weiterer Folge sollen nun die Ergebnissignale nicht nur nach ihrer Plausibilität bewertet, sondern auch jenen gegenübergestellt werden, die am realen System anhand von Testszenarien gewonnen wurden. Eine möglichst genaue Nachbildung der Ergebnisse soll in der Simulation am Entwicklungsrechner erreicht werden können. Bei Fehlfunktionen des B-Muster-BMS sollen Lösungen umgesetzt werden, die trotzdem plausible Simulationsergebnisse generieren können.
Ein wichtiges Ziel ist die Umsetzung von Hilfsmitteln, um die Benutzbarkeit der Simulationsumgebung am Entwicklungsrechner zu sichern und den Komfort während der Funktionsentwicklung zu erhöhen. Ob das Ziel, den Standard-Entwicklungsrechner als Simulationsumgebung verwenden zu können, erreicht werden kann, ist unter anderem zu ermitteln.
Die Simulationsumgebung soll die Möglichkeit bieten, die Zellparameter des BMS ohne großen Aufwand, nach den Vorgaben des Funktionsumfangs des Zellprototyps, manipulieren und diese automatisiert ins BMS einzubinden zu können. Ebenso soll auch die Möglichkeit geschaffen werden, dass generierte Signale der Simulationsumgebung mit Signal-Referenzen des Realsystems vom Prüfstand verglichen werden können. Wichtig dabei ist, dass der Benutzer dabei keinerlei Kenntnis über die zugrunde liegende Struktur besitzen muss, um die Parametersätze zu manipulieren, die Simulation zu starten und einen rudimentären Signalvergleich durchführen zu können. Somit stellt ein weiteres Ziel die einfache Handhabung der Simulationsumgebung dar, mit der Möglichkeit, den Funktionsumfang an die jeweiligen Anforderungen des Benutzers anzupassen.
Optional wird das Ziel gesetzt, eine am Prüfstand lauffähige Software aus einem

speziell definierten Parametersatz mittels Codegenerierungstool TargetLink® zu erstellen. Diese Software wird auf die BMU eines realen Systems übertragen um darauf einen speziellen Test zu durchlaufen, welcher beeinflusste Funktionalitäten ausreizen soll. Die gewonnenen Daten können als Referenz für das in der Simulation laufende, reduzierte Modell dienen.

2

Entwurf und Implementierung

2.1 Entwurf

2.1.1 Entwicklungsumgebung

Die Entwicklungsumgebungen am Simulationsrechner der Zell- und Funktionssteuerungsentwicklung wird aus der MathWorks MATLAB® Version 7.11.1 (R2010b) SP1 und der Simulink®-Toolbox gebildet, um eine modellbasierte Implementierung der gesamten BMU zur ermöglichen. Die durchschnittliche Leistung der Entwicklungshardware in der Zell- und Funktionssteuerungsentwicklung entspricht Intel Core i5 CPU Laptops mit rund 2,5 MHz und ca. 2 GB RAM Arbeitsspeicher. MATLAB/Simulink® wird auf dem Betriebssystem Microsoft Windows XP® in der 32 Bit-Version betrieben.

2.1.2 MATLAB

Die Steuerungssoftware der BMU, das BMS, wird mit Hilfe der Hardwareimplementierungsschnittstelle TargetLink® entwickelt, welche eine Umwandlung des modellbasierten MATLAB/Simulink®-Modells in den für das Realsystem notwendigen C – Code ermöglicht. In Simulink® werden dafür die Modellblöcke aus der TargetLink®-Library verwendet. Aus Kostengründen, die bei der Zusammenstellung und Lizenzierung einer Entwicklungsumgebung mit TargetLink® entstehen, wurden BMS – Module für die vorliegende Aufgabe (siehe Abschnitt 1.7.1) zur Verfügung gestellt, welche keine TargetLink®-Abhängigkeiten beinhalten. Aus diesen Modulen werden mit Hilfe der MATLAB® Toolbox RealTime-Workshop® S-Functions erstellt, welche die Funktionen ihrer entsprechenden Simulink®-Module übernehmen sollen, um die Start – und Simulationsdauer des BMS auf ein für der Praxis akzeptables Maß zu reduzieren. Für die Implementierung der im Folgenden beschriebenen Funktionen und die Simulation der Testfälle wurde der Umfang der standardisierten MATLAB/Simulink®-Installation und der voreingestellte MATLAB®-Editor verwendet. In dieser Arbeit kommen keine weiteren Toolboxes mehr zum Einsatz.

2.1.3 Modultypen

Wie bereits erwähnt, kann für die Simulation des BMS zwischen den TargetLink® freien Simulink®-Modulen oder ihren äquivalenten S-Functions gewählt werden. Die Simulink®-Module bieten den Vorteil der transparenten Darstellung und der Manipulation bzw. Erweiterung ihrer Funktionen, müssen aber bei jedem Simulationsstart des Modells erneut kompiliert werden. Dieser Vorgang kann je nach Komplexität oder der Anzahl der dafür benötigten Parameter oder LUTs einige Minuten in Anspruch nehmen. Dieser Zeitaufwand stellt bei der Entwicklung und der Simulation den großen Nachteil dieser Modulvariante dar. Ebenso müssen die benötigten Parameter dafür im Workspace[4] bzw. im selben Ordner zur Verfügung stehen. In weiterer Folge werden solche Module verwendet, wenn interne Signalverläufe beobachtet, oder deren Parameter manipuliert werden müssen.
Bei der Verwendung der S-Functions muss beachtet werden, dass ihre Konfiguration nicht verändert werden kann, es sei denn, sie bieten dafür Schnittstellen nach außen. Parameter, die deren Funktionalität charakterisieren, werden bei der Generierung mit Hilfe des RealTime-Workshops® an das Modul gebunden und können somit nicht nachträglich manipuliert werden. Dafür bietet diese Variante den großen Vorteil, den Kompiliervorgang nicht mehr ausführen zu müssen und somit die Zeitdauer

[4]Workspace: MATLAB® stellt für jede Sitzung einen reservierten Speicher zur Verfügung

der System-Initialisierung drastisch zu verkürzen. Ebenso verwendet der RealTime-Workshop® bei der Generierung der Module mathematische Optimierungsverfahren, mit denen die Simulationszeit selbst auch beschleunigt wird.
In der Praxis kann die vorher erwähnte Verhinderung der Modul-Transparenz auch einen Vorteil von S-Functions bedeuten. Bis auf die äußersten Schnittstellen kann somit die interne Funktionalität unbefugten Betrachtern verborgen bleiben.
Das TargetLink® freie BMS liegt in mehreren, nach Entwicklungsstand nummerierten, Versionen vor, die ihre eigenen Parameterlisten mit sich bringen. Beim möglichen Umstieg auf eine neuere BMS-Version muss darauf geachtet werden, dass die notwendigen Manipulationen mit wenig Aufwand durchgeführt werden können. In (siehe Abschnitt 1.7) wird die Notwendigkeit beschrieben, diverse Funktionserweiterungen vorzunehmen, um Manipulationen des Systems und Variationen der Betriebsbedingungen vornehmen zu können. Aus diesem Grund kann es vorkommen, dass die Verwendung von Simulink®-Modulen für so manche Funktionalität zwingend notwendig wird, bspw. im vorliegenden Fall das VCM (siehe Abschnitt 1.5.4.2). Die benötigten Funktionalitäten für die Manipulation des Modells und der anschließenden Simulation werden in weiterer Folge beschrieben.

2.1.4 Simulationsumgebung

Unter dem Begriff Simulationsumgebung wird im Rahmen dieser Arbeit ein Simulink®-Mdl[5] verstanden, dass als Basismodell für referenzierte Einzelmodule dient und eine Menge weiterer Funktionalitäten für die entwicklungsbegleitende Funktionsabsicherung zur Verfügung stellt. Im Modell namens Simulation_FRAME wird so eine Simulationsumgebung entworfen, welches das ZM als Signalquelle (siehe Abschnitt 1.5.2) beinhaltet und die einzelnen BMS-Funktionalitäten in eigenen Modulen referenziert.
Das Modell namens VCM_Mdl stellt ebenfalls eine Simulationsumgebung dar, die ihren Umfang auf die Funktionsanalyse des VCMs beschränkt.

2.1.4.1 Das Zellmodell

Der Entwicklungsfortschritt und die Parametrierung des ZMs liegen im Rahmen dieser Arbeit als B-Muster vor (siehe Abschnitt 1.3.6), werden aber in weiterer Folge nicht näher betrachtet. Es stammt aus der Abteilung Zellmodellierung bei MAGNA Steyr Fuel and Battery Systems zur Funktionsabsicherung früherer Projekte.
Die vom ZM generierten Signale werden in der vorliegenden Entwicklungsumgebung

[5] Abkürzung für Modell

in die entsprechenden Signalgruppen des BMS integriert, um dessen Simulation zu ermöglichen. Für die Simulation der in Serie geschalteten Zellen wird je ein Signal der kleinsten und der größten Zellspannung berechnet, um die interne, bautechnisch bedingte Spannungsstreuung zu simulieren. Zusätzlich erhält das BMS den Spannungspegel einer jeden Zelle im System, der periodisch, samt dazugehörigem Index, an das BMS übermittelt wird.
Eine wichtige Funktionalität des Zellmodells ist die optionale Simulation des internen Temperaturverlaufs. Per Startparameter kann gewählt werden, ob die Zelltemperatur von einem externen Signal oder, ausgehend von einem Starttemperaturwert, vom Modell berechnet wird. Sie ist von der Systembelastung und der eingestellten Kühlleistung abhängig, welche über die Durchflussmenge oder die Kühlleistung bestimmt werden kann. Die weiteren ZM-Funktionen werden in weiterer Folge nicht weiter betrachtet.

2.1.4.2 Simulation_FRAME

Der Simulation_FRAME beinhaltet auf höchster Ebene die Ansicht der Zusammenschaltung der benötigten Simulationskomponenten, im vorliegenden Fall das Zellmodell und das BMS. Zusätzlich findet man ein Subsystem mit den darin zusammengefassten Eingangssignalen, ein weiteres Subsystem, welches den rückgekoppelten Einfluss des BMS zum ZM steuert und ein Subsystem mit dem man den Einfluss der ZM-Signale umgehen kann (siehe Abb. 2.1).
Sollte der ZM-Einfluss für die gewünschte Simulation keine Rolle spielen, kann das Eingangssignal direkt als Input für das BMS hergenommen werden. Dafür muss lediglich die Konstante am Eingang des Subsystems (in weiterer Folge `InputSwitch` genannt) der Vorgabe entsprechend eingestellt werden (in Abbildung 2.1 wird zwischen Prüfstandsdaten (Source-Data) und ZM-Output als BMS-Input gewählt). Zur Beschleunigung der Funktionswahl wird eine Reihe von Schaltflächen entworfen, um die direkte Ausführung der Skripts und Funktionen mittels MATLAB®-Editor zu ersetzen. Die Schaltflächen dienen in dieser Arbeit der Simulationsinitialisierung, zur Darstellung von Ausgangssignalen, dem automatisierten Tausch zwischen den Simulink®-Modulen und den äquivalenten S-Functions, dem Zugriff zur Parameterreduzierung oder dem Laden eines Systemzustandes.
Die Simulation von ZM und BMS setzt eine plausible Bedatung der Signaleingänge und Startparameter voraus (siehe Abschnitt 2.1.4.4). Darin muss unterschieden werden, ob das Anlegen oder das Fernbleiben eines bestimmten Simulationssignals an einem bestimmten Port Einfluss auf die zu analysierende Systemfunktionalität hat.

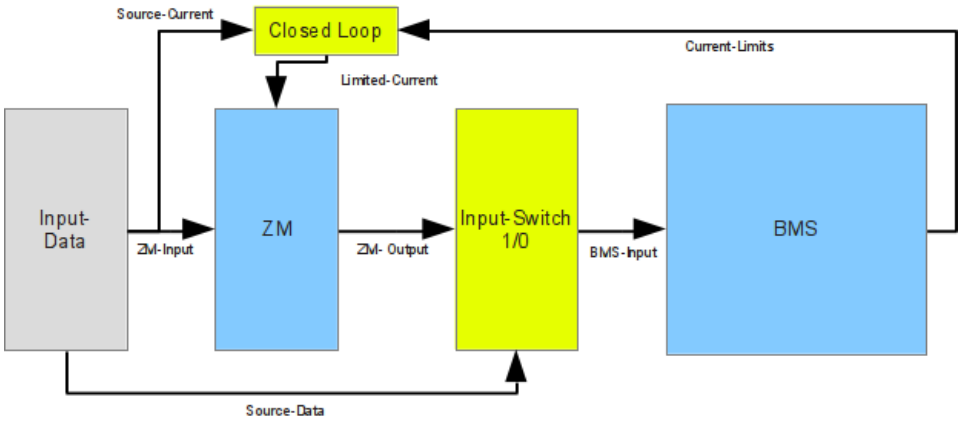

Abbildung 2.1: Blockaufbau des Entwurfs der Simulationsumgebung

Beispielsweise wird das Stromsignal des Sollstromes von der Simulation zwingend gefordert, hingegen stellt das Temperatursignal ein optionales Eingangssignal dar.

2.1.4.3 Isolierte-VCM-Simulationsumgebung

Aufgrund der Problemstellung aus (siehe Abschnitt 1.7) vorrangig die Parameter des VCMs zu manipulieren, wird das VCM als extrahiertes, eigenständig simulierbares Modul entworfen. Die Startzeit der Simulation bzw. die Dauer des Kompiliervorgangs kann mit einem reduzierten Umfang verkürzt werden. Die Idee hinter dem isolierten VCM ist die, bei mehrmaligen Parametervariationen und damit verbundenen Simulationsdurchgängen, die summierte Wartezeit zu verkürzen.
Dafür wird das Simulink®-Modell VCM_Mdl entworfen und im selben Ordner abgelegt. Da das VCM für eine Simulation eine Reihe von Eingangssignalgruppen benötigt, wird eine zusätzliche Methode im Simulation_FRAME eingefügt, welche alle Inputdaten des VCMs aufzeichnet (in weiterer Folge Signal-Monitor genannt) und diese dann dem VCM_Mdl als Signalquelle bereitstellt. Sie wird im integrierten VCM-Subsystem des BMS, an der Stelle wo die VCM-Module zusammengefasst werden, in Form eines Subsystems, welches die ins VCM eingehenden Signale zur Speicherung vorbereitet, eingebunden.

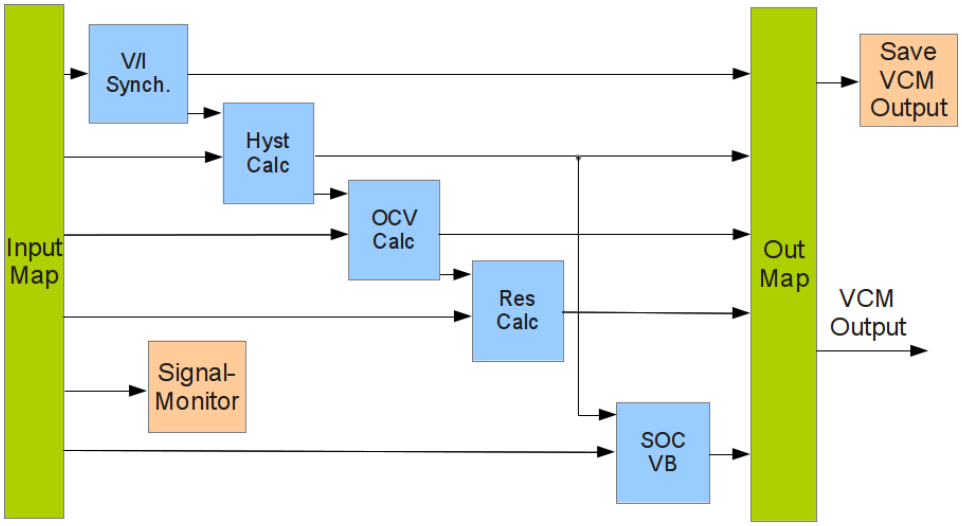

Abbildung 2.2: Die rot untermalten Modellblöcke sind im Vergleich zu Abb. 1.21 für die Signalaufzeichnung zuständig

Im Weiteren werden Basisbegriffe des Entwurfs der vorgestellten Simulationsumgebungen näher erläutert.

2.1.4.4 Initialisierung

Mit Initialisierung wird in diesem Rahmen der Ladevorgang, der für die Simulation benötigten Daten, in den MATLAB®-Workspace bezeichnet. Damit meint man den zu simulierenden Fahrzyklus mit den benötigten Signalen, den Parametern, welche die Funktionalitäten der Simulationsumgebung beeinflussen, Zustandswerte, die den Startzeitpunkt des Modells charakterisieren und die Stellungswahl der Simulink®-Switches, die auch während Simulation betätigt werden können, um den Verlauf zu beeinflussen. Bei der Verwendung der Simulink®-Module muss zusätzlich bei der Initialisierung darauf geachtet werden, dass dem Modell der entsprechende Parametersatz zugrunde liegt. Die Parametersätze unterscheiden sich zwischen den Modellversionen im Umfang und dem Entwicklungsstand. Das ZM steht in der vorliegenden Arbeit nur als Simulink®-Modell zur Verfügung, greift aber auf einen separaten Parametersatz zurück.
Eine wichtige Initialisierungsmaßnahme stellt die Bedatung jener LUTs dar, welche die Fehlerwerte gewisser Inputsignale in den Signalgruppenblöcken am Signaleingang

des BMS definieren. Sie beruhen auf Parametern, welche sich im vorliegenden Fall noch in Entwicklung befinden und deswegen während der Simulations-Initialisierung in aktuellster Form aus dem MKS®[6] geladen werden. Sollte keine Netzwerkverbindung bestehen, werden während der Initialisierung Erwartungswerte herangezogen. Betroffen davon sind die Temperatur- und Zellspannungssignale am BMS-Eingang bzw. Stromsensorparameter (siehe Abschnitt 2.2.2).
Am Ende des Initialisierungsvorganges wird eine Dialogbox eingeblendet, welche die wichtigen Start- und Simulationsinformationen auf einen Blick darstellt. Sie gibt Auskunft über jene Werte, die im Falle von fehlenden Informationen für eine korrekte Simulation manuell deklariert werden müssen. Betroffen sind davon der SOC, der Startwert der Hysterese und die Temperatur. Zusätzlich gibt die Box Auskunft über den Startzeitpunkt, um auf einen Offset[7] hingewiesen zu werden, oder den Endzeitpunkt, falls die Simulationsdauer das Speichervermögen des Rechners möglicherweise überschreitet (siehe Abschnitt 2.3.2.2).
Die Initialisierung des VCM_Mdls findet analog statt, mit dem Unterschied, dass durch die Signalaufzeichnung des Monitors im BMS keine weiteren Schritte mehr notwendig sind, da im Grunde nur komplette Eingangsdatenpakete simuliert werden.

2.1.4.5 Startparameter

Während eines korrekten Bootvorganges, unmittelbar vor dem Gebrauch eines realen Batteriesystems, muss die Software Zustandsinformationen des letzten Betriebes aus dem NVM laden (siehe Abschnitt 1.5.1)[MSB12]. Für die Simulation von Fahrzyklen hängt diese Funktionalität von den Simulationsanforderungen des Benutzers ab. Der Zugriff der Startzustände muss deshalb transparent und manipulierbar gehalten werden.
Benötigt wird pro Simulationslauf der komplette Umfang der NVM-Startwerte, bei fehlenden oder ungültigen Werten greift das Modell auf Default-Einträge zurück. Eine Ausnahme stellt der SOC dar, der im Fehlerfall eine Alternative bereit stellt (siehe Abschnitt 1.5.4.3), aber bei Ausfall dieser Funktion ebenfalls Default-Werte verwendet.
Die Handhabung des modellierten NVMs wird in drei Varianten entworfen:

1. Für die erste existiert ein Abschnitt im Initialisierungsskript für die Deklaration der Werte (siehe Abschnitt 2.2.2.1).

[6] MAGNA Steyr verwendet zur entwicklungsunterstützenden Datenverwaltung die Software MKS®
[7] Sollte der momentane Fahrzyklus einer von mehreren, zusammenhängenden Teilen sein, kann der Anfangszeitpunkt von Null abweichen

2. Die zweite Variante stammt aus dem vorangegangenen Projekt, in der auch zur Simulationszeit innerhalb vom Simulink®-Modell auf Soll-Werte umgeschaltet werden kann. Verwendung findet diese Manipulationsmöglichkeit z.B. beim Erzwingen eines vom tatsächlichen Wert abweichenden Start-SOCs.
3. Unter den Schaltflächen im Simulation_FRAME findet sich die dritte Variante, den NVM zu initialisieren, in dem der Endzustand eines auswählbaren Fahrzyklus für die Weiterfahrt ausgewählt wird. Voraussetzung ist dafür, dass dieser Fahrzyklus im Simulation_FRAME bereits simuliert und erfolgreich abgeschlossen wurde.

Das VCM_Mdl greift auch auf die NVM-Startwerte beim Simulationsstart zurück, jedoch kann deren Bedatung aufgrund der Signalaufzeichnung des Signal-Monitors vernachlässigt werden. Trotzdem besteht die Möglichkeit, gleich wie im Simulation_FRAME, die Startwerte manuell variieren Werte zu können.

2.1.4.6 Simulationsstart

Der Start der Simulation erfolgt über die *Start-Simulation*-Funktion von Simulink®, ohne weitere Maßnahmen berücksichtigen zu müssen. Ausgenommen bei der in (siehe Abschnitt 2.2.4) beschriebenen Simulationsmethode, wo ein Skript gestartet wird, welches die Initialisierung und den Simulationsstart automatisch abarbeitet. Anschließend findet der Kompiliervorgang statt, welcher für eine zwischenzeitliche Komplettauslastung des verwendeten Entwicklungsrechners sorgt.

2.1.5 Simulationsdaten

Der weitere Ablauf der Simulation wird vom Profil der Quell- oder Eingangssignale bestimmt. In manchen Fällen müssen diese Signale für die Simulation vorbereitet werden. In allen Fällen wird eine Anpassung des generierten Simulationsergebnisses an den jeweiligen Analysebedarf nötig sein, wie die folgenden Abschnitte beschreiben.

2.1.5.1 Eingangssignale

Eingangssignale sind im Grunde jene Signale, die während eines Testlaufes des realen Systems am Prüfstand aufgezeichnet werden und somit den in dieser Arbeit oft genannten Fahrzyklus bilden. Die Software am Prüfstand stellt eine Reihe ihrer Eingangs- und Ausgangssignale über den CAN-Bus zur Verfügung, zum einen

als Steuerungssignal mehrerer Systeme im Fahrzeugbetrieb, zum anderen als Zusatzinformation für den Entwickler. Die Bereitstellung der Eingangssignale für die Simulationsumgebung wird in drei Varianten entworfen:

1. Die Signale liegen innerhalb des Prüfstandsdaten-Pakets in Gruppen vor, aufgrund unterschiedlicher Aufzeichnungsraten in jeder Gruppe mit eigenem Zeitvektor. Bei weiterer Verwendung innerhalb von Simulationsanwendungen wie MATLAB® sollte aber immer auf den Wert der Aufzeichnungsrate geachtet werden. In manchen Fällen kann sie je nach Signalgruppe variieren. Der Grund dafür liegt in der Reduzierung der Datengröße, die bei Signalen mit geringer Änderungsrate über die Zeit, wie z.B. der Systemtemperatur, durchaus angewendet werden kann.
   Der Verlauf der Systemtemperatur kann optional in Form eines vorgegebenen Eingangssignals, oder unter Verwendung eines Temperaturmodells simuliert werden. Letzteres berechnet das Temperaturverhalten der Zelle, ausgehend von einem vorgegebenen Starttemperatur-Wert (siehe Abschnitt 1.5.2). Das BMS benötigt die Temperaturinformation für die Simulation, unabhängig von welcher Quelle sie stammt.

2. Sollten weniger komplexe Testfälle benötigt werden, wird die Möglichkeit entworfen, per MATLAB®-Skript einfache Fahrzyklen zu generieren, die in einfachster Form, neben den wichtigen Startparametern, nur aus dem Stromvektor bestehen. Das Skript kann in nur wenigen Schritten eine plausible Simulationsbasis generieren, die sich in den Quelldatensatz der übrigen Fahrzyklen eingliedert. Es unterliegt aber der Verantwortung des Benutzers, Startparameter wie z.B. jene der Hysterese-Information mit dem des Start-SOCs zu kombinieren, sodass eine plausible Startbedingung zu Stande kommt.

3. Als zusätzliche Möglichkeit, dem Modell ein Stromprofil zur Verfügung zu stellen, wäre der Signal-Generator, der von MATLAB/Simulink®-Library zur Verfügung gestellt wird. Ein sehr einfaches Stromprofil kann entweder direkt mit den gegebenen Werkzeugen oder, etwas komfortabler, in einer Microsoft Excel® Arbeitsmappe erstellt werden. In der ersten Spalte muss lediglich der Zeitvektor, in der zweiten der Stromvektor vorgegeben werden. Die in Excel® zur Verfügung stehenden Tools zur Generierung eines Stromprofils können die Arbeit in gewissen Anforderungen erheblich erleichtern, z.B. mit Hilfe der Autovervollständigung. Die Bedatung der Startparameter unterliegt den Simulationsabsichten des Benutzers. In diesem Fall wäre es denkbar, einen Fahrzyklus mit den gewünschten Startparametern aus den ersten beiden Varianten zu laden und ihnen das per Generator erstellte Stromprofil hinzuzufügen.

Bei allen drei Varianten zur Generierung der Simulationsdaten muss der am Entwicklungsrechner zur Verfügung stehende Arbeitsspeicher im Auge behalten werden (siehe Abschnitt 2.3.2.2). Besteht die Möglichkeit, dass der verfügbare Speicher für die Dauer der Simulation nicht ausreichen sollte, wird der Benutzer während der Initialisierung darauf hingewiesen.

2.1.5.2 Aufteilung von lange andauernden Fahrzyklen

In der Praxis werden die Tests am Prüfstand über mehrere Stunden gefahren, was bei einer entsprechend hohen Anzahl der aufzuzeichnenden BMS-Signale unter der Voraussetzung der unveränderbaren, schnellen Abtastzeit (siehe Abschnitt 2.3.1.1), zu Speicherüberschreitungen führen kann. Es ist zwar möglich, am Prüfstand die Testaufzeichnung in kleinere Pakete zu zerteilen, trotzdem kann bei entsprechend hohem Datenaufkommen während der Simulation eine Speicherknappheit entstehen. Um trotzdem ohne großen Aufwand die späteren Zeitpunkte verschiedener Zyklen simulieren zu können, wird eine Methode entworfen, die Datensätze in kleinere, simulierbare Pakete zu teilen (siehe Abschnitt 2.2.4). Dem Entwickler steht es in diesem Fall frei, in wie viele Teile er einen Zyklus unterteilt. Die Simulationsergebnisse werden chronologisch durchnummeriert abgespeichert, um in weiterer Folge dargestellt werden zu können (siehe Abschnitt 2.1.5.3). Für die korrekte Darstellung ist es aber in jedem Fall notwendig, die bereits vergangene Zeitdauer als Zeit-Offset mitzuspeichern bzw. den Zustand am Ende der Simulation zugänglich zu machen.

2.1.5.3 Darstellung der Simulationsergebnisse

Der Inhalt des Simulations-Ergebnisses soll für den jeweiligen, speziellen Schwerpunkt der Funktionsanalyse, alle benötigten Signale über die geforderte Zeitdauer beinhalten (siehe Abschnitt 2.4). Aufgrund der hohen Komplexität der BMS-Funktion kann im Rahmen dieser Arbeit nur auf die Darstellung der gebräuchlichsten, aussagekräftigsten bzw. die aus der Aufgabenstellung (siehe Abschnitt 1.7.1) notwendigen Signale Rücksicht genommen werden. MATLAB/Simulink® stellt für die Signalaufzeichnung mehrere Werkzeuge zur Verfügung, welche in der vorliegenden Arbeit für die verschiedenen Fälle der Signalaufzeichnung variiert werden müssen. Die Unterschiede betreffen den Ort der Datenablage, also im Workspace oder in einer gespeicherten Datei, die Möglichkeit Signalgruppen samt Simulationszeit zusammenzufassen oder den Signalverlauf bereits während der Simulation optisch darstellen zu können.
Für die Analyse von Modellfunktionalitäten ist es in vielen Fällen erforderlich, die

generierten Ausgangssignale der Simulation mit den Äquivalenten des Testlaufes am Prüfstand zu vergleichen. Dabei muss aber bei der automatisierten Darstellung der Ergebnisse geachtet werden, ob die entsprechende Vergleichsreferenz überhaupt im Prüfstandsdatensatz verfügbar ist (siehe Abschnitt 2.3.2.5). Sollte sie nicht verfügbar sein, kann der Grund entweder im Erstellen des Tests per Skript nach Fahrprofilvariante 2 (siehe Abschnitt 2.1.5.1), oder in einem veränderten Datensatzumfang liegen. Letztere muss nämlich vor einem Test am Prüfstand definiert werden, da vor allem jene Signale, die für die Funktionsverifikation notwendig sind, in hoher Anzahl zur Verfügung stehen. Mit der Zahl der Signale steigt auch der benötigte Speicherbedarf des Testlaufes am Prüfstand, der aber aus Speichergründen gering gehalten werden muss.
Ein wichtiger Punkt beim Vergleich der Simulationsergebnisse mit den äquivalenten Referenzsignalen vom Prüfstand ist die Kontrolle der Softwareversionen beider Seiten. Bei Abweichung der Versionsnummern vom BMS-Modell und der Prüfstandsoftware können manche Signalverläufe voneinander abweichen, bspw. wegen veränderter Parameter nach der Weiterentwicklung einer Funktion.
Besonderes Augenmerk wird bei der Darstellung der Simulationsergebnisse auf die Funktionen SOC und SOP gelegt, da deren Berechnung von den in (siehe Abschnitt 1.5.1) genannten, fehlenden Batteriemodulen in der Simulation am Entwicklungsrechner wenig bis kaum beeinflusst werden. Das ZM liefert den nötigen Input in Form von Strom-, Spannung- und Temperatursignalen, um dem BMS die Zustandsberechnung zu ermöglichen. Für die Validierung der vom BMS generierten Ausgangssignale stehen zwei Möglichkeiten des Signalvergleichs zur Verfügung. Entweder, es werden die vom ZM eigenständig berechneten Zustände (z.B. SOC und OCV) herangezogen, oder es werden die am Prüfstand generierten Ausgangssignale in die Darstellung geladen.

2.1.6 Parametrierung des BMS

Für die TargetLink®-Version des BMS steht zur Parameterverwaltung und –speicherung die Datencontainer-Struktur *Data Dictionary* von dSpace® zur Verfügung. Aus den gespeicherten Informationen wird bei der Generierung des C-Codes für den Betrieb auf der Systemhardware mittels TargetLink® zurückgegriffen. Das Verfahren der Codegenerierung setzt TargetLink® und *Data Dictionary* voraus.
In der vorliegenden Arbeit kann jedoch nicht auf die TargetLink® und *Data Dictionary* Unterstützung zurückgegriffen werden, wie in (siehe Abschnitt 1.7.1) begründet. Die Systemfunktionalitäten stehen in Form der Standard-Simulink®-Library zur Verfügung. Die Parameter dieser BMS-Module und jene des ZMs werden in Form

von MATLAB®-Files im selben Arbeitsverzeichnis gespeichert.
Die Parameter des BMS werden in Form von Skalaren, Vektoren oder Matrizen in einer MATLAB®-Struktur mit dem Namen `parameter_const_list` eingebundenen und gespeichert. Die Struktur gibt Auskunft über diverse Eigenschaften wie den Typ, die Kategorie, Grenzwerte oder die Dimensionen des Parameters, die in Form von Konstanten, Zuständen und LUTs gespeichert sind. Wird eine dauerhafte Manipulation solcher Werte gefordert, muss auf diese Struktur zurückgegriffen werden. Bei Bedarf ist es möglich, auf Basis dieses reduzierten Parametersatzes eine am Prototypensystem lauffähige Software zu generieren, worauf aber in weiterer Folge nicht näher eingegangen wird.
Die Parametrierung der Simulink BMS-Module selbst findet auf drei Arten statt:

1. In der ersten Variante wird die Bedatung in Form eines direkt eingestellten Werts im Simulink-Block durchgeführt. Diese eher unpraktische Implementierung entstammt der Konvertierung aus den TargetLink®-Modulen.

2. In der zweiten Variante werden die Daten in eine, bei der Initialisierung erstellten Struktur, im Workspace abgelegt. Diese Struktur muss in diesem Fall immer neu angelegt werden und kann ohne weitere Maßnahmen nicht für weitere Simulationsanforderungen weiterverwendet werden, bietet aber den Vorteil einer schnellen Zugriffsmöglichkeit, auch zur Laufzeit der Simulation. Der Performancegewinn wird bei der Bedatung der LUTs anhand guter Laufzeit-Ergebnisse erkennbar (siehe Abschnitt 2.4).

3. Die eigentliche, *Data Dictionary* ersetzende Variante, die Module zu bedaten, stellt zugleich die aufwändigste Variante dar. Sie basiert auf einer, aus vergangenen Projekten stammenden Funktion `ddv`, welche die Bedatungsfunktion von *Data Dictionary* in Verbindung mit TargetLink® ersetzt (siehe Abschnitt 2.2). Bis auf wenige Ausnahmen sind in der vorliegenden BMS-Version alle Simulink-Blöcke auf diese Weise bedatet.

2.1.6.1 Reduktion der Parameter

Die Parameterreduktion erfolgt im Simulation_FRAME wie auch im VCM_Mdl auf dieselbe Weise. Wichtig beim Benutzen der Funktionalität ist die Tatsache, dass nur Simulink-Module auf die `parameter_const_list` zugreifen und somit in ihrer Parametrierung manipulierbar sind.
Welche Parameter letzten Endes in welcher Form manipuliert werden, soll beliebig ausgewählt werden können. Somit werden alle in Frage kommenden Stützstellen und Grenzen in einer graphischen Auswahlmaske dargestellt. Unterschieden wird dabei

in der Aus- bzw. Abwahl bestimmter Stützpunkte, bspw. beim SOC, den Stromgrenzen oder der Eingabe von absoluten Grenzwerten, wie bei der Systemtemperatur. Ebenso soll der Name des daraus generierten Ergebnisses schon vorher editierbar sein, um eine auf die Manipulation hinweisende Bezeichnung erhalten zu können.
Die laut (siehe Abschnitt 1.7.1) gewünschten Funktionseinschränkungen, um in weiterer Folge die Dauer der Zellcharakterisierung reduzieren zu können, werden mit Einschränkungsmaßnahmen in den Bereichen SOC/OCV, Belastungsstrom, Innenwiderstand und dem Betriebstemperatur-Bereich erreicht.
Das Ergebnis der Parameterreduktion stellt eine neue `parameter_const_list` dar, in der die vorher schon vorhandenen LUTs in ihrer ein-, zwei- oder dreidimensionaler Form, im Umfang verändert wurden. Die Dimensionen der LUTs müssen dabei unverändert bleiben, da jede Dimension für eine Eingangsgröße steht, die trotz Reduktion der Stützstellen gültige Werte bereitstellen muss.
Der Vorteil liegt darin, dass alle Zustandsoperationen des BMS, die auf eine im Parameterumfang reduzierte LUT aus der `parameter_const_list` zugreifen müssen, ohne weitere Maßnahmen nun eingeschränkt funktionieren. Nachteilig wirkt sich der Umstand aus, dass dabei keine Informationen über den ursprünglichen Zustand der LUTs in der `parameter_const_list` behalten werden. Deshalb muss eine Maßnahme geschaffen werden, vor jeder Parameterreduktion auf einen unveränderten Parametersatz zugreifen zu können, ohne den Einfluss vorhergegangener Arbeitsschritte.

2.1.7 Laufzeitverbesserung der Simulation

Aus Abschnitt 1.7 geht hervor, dass die Simulationsumgebung für den Betrieb am Entwicklungsrechner optimiert werden soll. Der Entwurf solcher Optimierungen folgt an dieser Stelle:

2.1.7.1 Tausch der Module

Der Entwurf sieht vor, dass über zwei Schaltflächen im Simulation_FRAME zwischen der Verwendung der Simulink-Module und der S-Functions getauscht werden kann. Wichtig dabei ist die korrekte Bezeichnung der BMS-Version, welche nach dem Aufruf abgefragt wird. Sie muss im Arbeitsverzeichnis (siehe Abschnitt 2.2.1.1) als Simulink- und als S-Function-Variante verfügbar sein.

2.1.7.2 Tausch der LUTs

Die bereits existierenden 3D-LUTs im vorliegenden B-Muster BMS-Modell werden von externen Softwareanbietern zur Verfügung gestellt. Sie basieren auf einer speziellen Implementierung, welche Schnittstellen für die Codegenerierung per TargetLink® möglich macht. Die Struktur bilden Pre-Lookup-Tabellen, welche die Bedatung für den nachfolgenden Interpolation-3D Block liefern.
Bei der Konvertierung in das von TargetLink® befreite BMS-Modell werden dieselben Strukturen zwar übernommen, nur erfolgt die Bedatung der LUTs mit mehreren, aus den in (siehe Abschnitt 2.1.6) vorgestellten Varianten. Für die Problemstellung (siehe Abschnitt 1.7.1) gilt diese Form als unpraktisch und ist deswegen in weiterer Folge ungeeignet.
Aus diesem Grund werden die LUT-Strukturen gegen Subsysteme getauscht, in denen die Standard-Simulink®-Pendants Anwendung finden. Diese LUTs werden mit der performanten Methode 2 aus Abschnitt 2.1.6 bedatet. Ein positiver Nebeneffekt dieses Schritts ist eine erhebliche Verkürzung der Dauer des Kompiliervorgangs vor einer Simulation, da das Initialisieren und Kompilieren der teils umfangreichen 3D-LUTs in ihrer ursprünglichen Form generell einen Großteil der Startzeit in Anspruch nimmt.

2.2 Implementierung

Die Umsetzung des Entwurfs (siehe Abschnitt 2.1) nach der Aufgabenstellung (siehe Abschnitt 1.7.1) integrierte über die Dauer dieser Arbeit hinweg, nacheinander eine Reihe von steigenden B-Muster BMS-Versionen und eine B-Muster ZM-Version.
Die in (siehe Abschnitt 2.1.6) erwähnte MATLAB®-Struktur `parameter_const_list` beinhaltet in zusammengefasster Form Parameter, Konstanten, Zustände und Simulationseinstellungen und bedatet jede einzelne BMS-Version ihrem Entwicklungsstand entsprechend. Diese Struktur sollte nicht mit der Parameterdatei des ZMs verwechselt werden, da die Zelldaten von der eigentlichen Aufgabe (siehe Abschnitt 1.7.1) unangetastet bleiben sollen. Die im Folgenden beschriebenen Funktionen und Skripte sind alle im selben Ordnerverzeichnis des Projektes zu finden.
In der `parameter_const_list` ist ebenfalls die Variable für die globale Abtastzeit von $20ms$ unter `CycTime` zu finden. Während der Umsetzung muss dieser Wert eingehalten werden.
Voraussetzung für die korrekte Bedatung der Simulink®-Bauteile mit Konstanten und Parametern ist die Funktion `ddv` im selben Ordnerverzeichnis, bzw. referenziert in einem der MATLAB®-Pfade. Sie ist dafür verantwortlich, dass die korrekten Werte der Parameter und der Konstanten, in die Wertefelder der jeweiligen TargetLink®-Pendants aus der Simulink®-Library, eingetragen werden können. Dafür wird im Wertebereich des zu bedatenden Simulink®-Bauteils die Funktion mit dem entsprechenden Übergabe-Parameter, direkt aus der `paramter_const_list`, eingetragen.

Simulation_FRAME: die höchste Modellebene der Simulationsumgebung wurde in Abschnitt 2.1.4.2 vorgestellt. Für die bessere Übersicht befinden sich die Modellblöcke der BMS-Eingangssignalgruppen eine Ebene darunter, in jenem Subsystem, in dem das vereinfachte Interface des BMS zu sehen ist.
Der `InputSwitch` (siehe Abschnitt 2.1.4.2) steht deswegen auf höchster Ebene, da sein eingestellter Wert sofort ersichtlich sein soll bzw. in manchen Fällen eine schnelle Umschaltung notwendig wird. Der Wert des Schalters muss 1 für die ZM-Daten und 0 für die Simulation mit Prüfstandsdaten sein, kann aber auch als Reservierung eines Parameters für zukünftige Automatisierungen betrachtet werden.
Die Rückkopplung der Stromlimitierungen stellt ein zusätzliches Feature bei der Verwendung des ZMs dar, damit unabhängig vom verwendeten ZM, die SOP-Begrenzungen in das Stromprofil des Fahrzyklus einfließen. Der Einfluss der Rückkopplung im Simulationsverlauf wird per `Manual-Switch` im `Closed-Loop`-Block aktiviert.

Der `Input-Block` beinhaltet neben dem Stromprofil auch eine Reihe weiterer Flags und Startparametern, welche die Funktionalität des ZMs zu- oder abschalten. Dies stellt eine Notwendigkeit dar, sollte es als S-Function ins Modell eingebunden werden. Darauf wird aber in weiterer Folge keine Rücksicht mehr genommen (siehe Abschnitt 2.1.4.4).

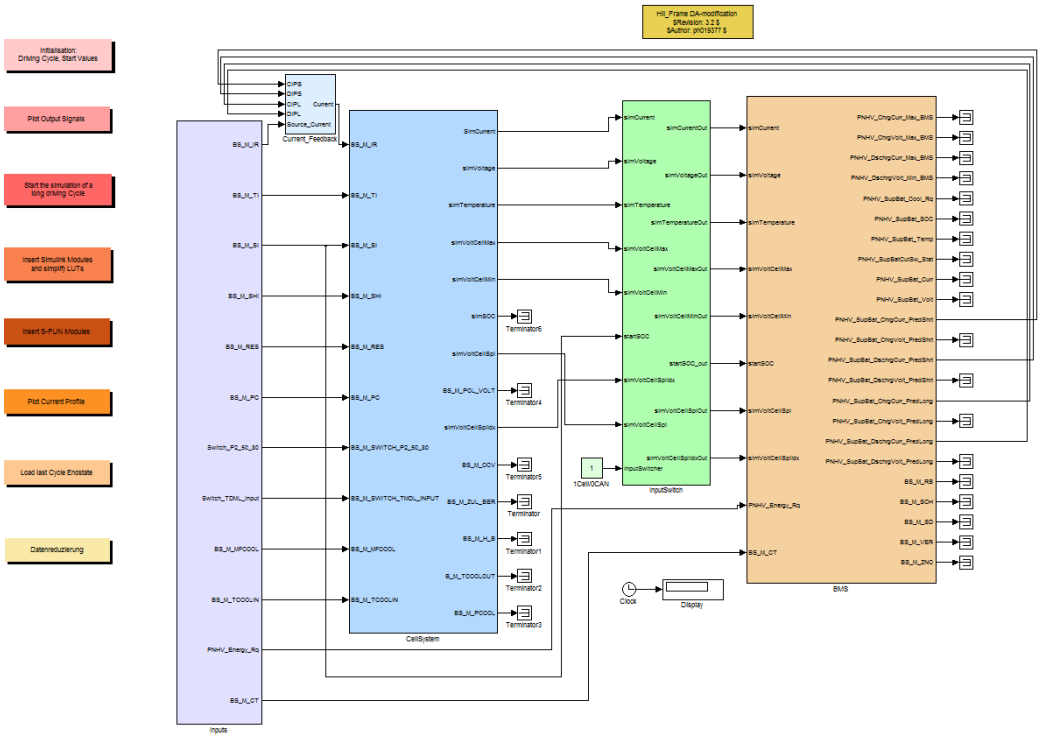

Abbildung 2.3: Die höchste Ebene des Simulation_FRAMEs. Zu sehen sind die Schaltflächen, der Input-Block, die Rückkopplung, das ZM, der `Input-Switch` und das BMS (von links nach rechts)

Im BMS-Subsystem ist die BMS-Eingangssignal-Generierung zu sehen, die je nach Signalgruppe in getrennten Blöcken stattfindet. Die Signale werden entweder vom ZM generiert, aus den Prüfstandsdaten gelesen oder aus den in Abschnitt 2.2.2 referenzierten Quellen mittels LUTs gebildet. Das Laden bzw. die Sicherung der NVM-Werte stellt eine wichtige Funktion dar und wird deshalb in Abschnitt 2.2.2.2 näher beschrieben. Die weiteren Komponenten dieser Simulink®-Ebene werden in dieser Arbeit nicht benutzt und deshalb außer Acht gelassen.
Ein weiteres Merkmal in dieser Simulationsumgebung ist das VCM-Signal-Monitoring, welches zu dem Zweck entworfen wurde, um im VCM_Mdl den vollständigen

Fahrzyklus simulieren zu können, der aber als Input die aufgezeichneten Signale benötigt, die sonst in keiner Form erstellt werden können (siehe Abschnitt 2.1.4.3). Die Eingangssignale des VCMs werden dafür, ihren Signalgruppenanordnungen entsprechend, in Simulink®-Scopes geleitet und während der Simulationszeit im Workspace abgelegt. Abschießend werden die erzeugten Signal-Strukturen mit den restlichen Ergebnissen der Simulation gespeichert (siehe Abschnitt 2.2.4.1).

VCM_Mdl: Im VCM_Mdl wurde ein Simulink®-Subsystem implementiert, welches nur das VCM-Modul (siehe Abb. 1.20) beinhaltet und die benötigten Schnittstellen zu den außen liegenden Signalgruppenquellen bereitstellt, die aus dem Ergebnis des Signal-Monitors bedatet werden.

Beide Modelle besitzen eine Reihe von Schaltflächen für den erleichterten Zugriff auf diverse Funktionen, wie z.B. Initialisierung, Ergebnisdarstellung oder Parameterreduktion. Die Schaltflächen sind inhaltslose Simulink®-Subsysteme, die optisch über die Masken-[8] und Farbeinstellungen editiert wurden und per OpenCallback die gewünschte Funktion ausführen.

2.2.1 Vorbereitung

2.2.1.1 Ordnerstruktur

Für den automatisierten Betrieb des Simulation_FRAMEs muss die im Folgenden beschriebene Ordnerstruktur vorhanden sein. Die zu simulierenden Fahrzyklen, ob am Prüfstand aufgezeichnet oder per Skript erstellt, liegen im Verzeichnis des Projekts im Ordner /Data. Die Skripts zur Generierung einfacher Fahrzyklen liegen in einem eigenen, beliebigen Ordner, der sich aber für die Automatisierung ebenfalls im selben Arbeitsverzeichnis befinden muss. Nach der ersten Simulation wird der Ordner /Results, mit den Simulationsergebnissen als Inhalt, automatisch angelegt, welcher für die weitere Signaldarstellung benötigt wird.
Ebenfalls auf dieser Ebene müssen die versionsspezifischen Rohmodelle des BMS im entsprechend bezeichneten Ordner abgelegt werden. Von jeder Version existieren Rohmodelle vom Simulink®- und vom S-Function-BMS. Zusätzlich liegt in diesem Ordner die zur BMS-Version zugehörige `parameter_const_list`, um für die Simulink®-Module die Charakteristik zu liefern.
Der einzige, ausgelagerte Ordner mit dem Namen /cm_parameter_determination,

[8] Im Feld für die „Icon drawing commands“ wird mit dem Befehl `disp('Beschriftungstext');` die Beschriftung des Subsystems ermöglicht

liegt eine Ebene über dem Arbeitsverzeichnis. Sein Inhalt wird für die Stützstellenbestimmung, im Rahmen der Parameterreduzierung, herangezogen.

2.2.1.2 Zell Modell

Um den Aufwand beim Austausch des ZMs (z.B. gegen eine neuere Version) zu minimieren, wurden nur wenige Änderungen während der Umsetzung der Simulationsumgebung vorgenommen. Somit wird das Start-Skript des ZMs (`StartCellModel_basic_CI`), welches für ZM-spezifische Simulationen erstellt wurde, weiterhin aufgerufen. Jedoch werden dessen Startwertdefinitionen und Signalbedatungen von der nun globalen Systeminitialisierung der Simulationsumgebung übernommen (siehe Abschnitt 2.2.2.1). An dieser Stelle könnte auch die Signalabtastung des ZMs modifiziert werden, wird aber aufgrund der in (siehe Abschnitt 2.3.1.1) beschriebenen Einschränkung des BMS auf $2ms$ belassen.
Bei der Parameterdefinition des ZMs muss lediglich in dessen Parameterdatei `SetCellModelData_CI` der Modul-Pfad[9] des ZMs in der Simulationsumgebung angepasst, oder im Falle von Erweiterungen, editiert werden. Das Modell selbst wurde um den Signaleingang des Temperaturvektors erweitert, um die beschriebene Option zur Temperaturberechnung zu ermöglichen. Dieser Signaleingang wird in der aktuellen ZM-Version ansonsten nicht verwendet.
Für die Funktionsanalyse (siehe Abschnitt 2.4) war es außerdem notwendig, eine Signalaufzeichnung in Form eines Scopes in das Modell zu integrieren. Die hiermit aufgezeichneten Daten stellen zum einen die Eingangsdaten des BMS dar, zum anderen kann z.B. der SOC oder der OCV des ZMs als Referenzsignal für das BMS dienen.

In den folgenden Abschnitten werden die entworfenen Funktionalitäten umgesetzt und in die Simulationsumgebung eingebunden. Hauptaugenmerk wird dabei auf die Optimierung der Simulationsperformance und die Funktion der Parameterreduktion gelegt.

[9] Um Simulink®-Module mit Hilfe von MATLAB®-Befehlen zu manipulieren, muss dem Befehl der komplette Pfad durch die Subsysteme, bis zum betroffenen Bauteil hin, bekannt sein

2.2.2 Eingangssignale

Um die Eingangssignale in der Simulation nutzen zu können, wurde eine Methode bevorzugt, in der die `Repeating_Sequence`-Blöcke aus Simulink® zum Einsatz kommen. Sie verknüpfen das Datensignal mit dem zugehörigen Zeitsignal zu einer Periode, was den Vorteil mit sich bringt, dass unabhängig von der Endzeit der Simulation, das Signal periodisch geladen werden kann.
Verwendet wird das `Repeating_Sequence`-Bauteil bei der Strom-Bedatung des ZMs bzw. im Alternativfall bei folgenden Signalen:

- Current: Der Belastungsstrom
- Voltage: Die Systemspannung
- CellVoltMin: Die minimale Zellspannung zum jeweiligen Messzeitpunkt
- CellVoltMax: Die maximale Zellspannung zum jeweiligen Messzeitpunkt
- Temperature: Die Systemtemperatur
- CellVoltSpl: Eine aus dem System herausgelöste, einzelne Zellspannung
- CellVoltSplIdx: Der System-Index der herausgelösten Zellspannung

Sollte das ZM mittels `InputSwitch` in die Simulation einbezogen werden, ist es für die Generierung dieser, für die Simulation notwendigen, BMS-Eingangssignale verantwortlich.
In beiden Fällen der Modellsimulation muss beachtet werden, dass manche der benötigten Signale in keiner Form im Fahrzyklus-Datenpaket verfügbar sind (siehe Abschnitt 2.1.5.1). Betroffen sind die Signalfehler der anliegenden Zellspannung, der Systemspannung und der Temperatur (siehe Abschnitt 1.5.3.6). Sie werden in der vollständigen Software am Prüfstand, von den jeweiligen Modulen der betreffenden Sensoren generiert und an das BMS gesendet. Die Simulation hingegen benötigt eigene, eindimensionale LUTs, die im jeweiligen Eingangssignal-Block sitzen und zur Laufzeit aus der entsprechenden Signal-Basis die zugehörigen Signalfehler generieren.
Ein wichtiger Schritt der Implementierung war die Sicherstellung, dass diese LUT-Parameter bei der Systeminitialisierung aus dem MKS®-Ordner der jeweiligen Sensor-Software geladen werden. Somit stehen immer die aktuellen LUT-Werte für die Simulation zur Verfügung.
Dafür wird das Skript `initializeInputLuts` aufgerufen, welches die Generierung der `lutdef_list`, mit Hilfe des jeweiligen Sensor-Skripts aus dem MKS®, ermöglicht. Die Offset-Werte der Systemspannungssensoren werden von einem weiteren

Skript aus dem MKS® generiert, wobei eine `parameter_const_list` entsteht. Sie trägt aus Entwicklungsgründen denselben Namen wie die Parameterstruktur des BMS, darf aber an dieser Stelle nicht damit verwechselt werden. Die Werte der Signalfehler werden anschließend aus dem aktuellen Spannungssignal, in Abhängigkeit eines Faktors und eines Offsets, berechnet.
Im Rahmen der Stromsignalgruppe müssen zwei Arten von Zusatzsignalen generiert werden:

1. Für die Toleranzsignale des Stromsignals werden ein Verstärkungsfaktor und ein Offset mit Hilfe eines dritten Skripts aus dem MKS® generiert. Die erhaltenen Toleranzwerte und der Offset werden anschließend zur Laufzeit zum Eingangsstromsignal addiert oder subtrahiert und die somit neu generierten Signale der Strom-Signalgruppe hinzugefügt.

2. Die Signalfehler des Stromsensors müssen laut BMS-Design zu jedem Zeitpunkt kleiner bleiben, als die Toleranzsignale, also von ihrer Wertespanne eingeschlossen werden. Um diese Voraussetzung zu garantieren, wird zur Vereinfachung der Problematik, die Hälfte der Differenz von Toleranzwert zu Stromwert als Aufschlag des Min- und Max-Signalfehlers verwendet (siehe Abb. 2.4).

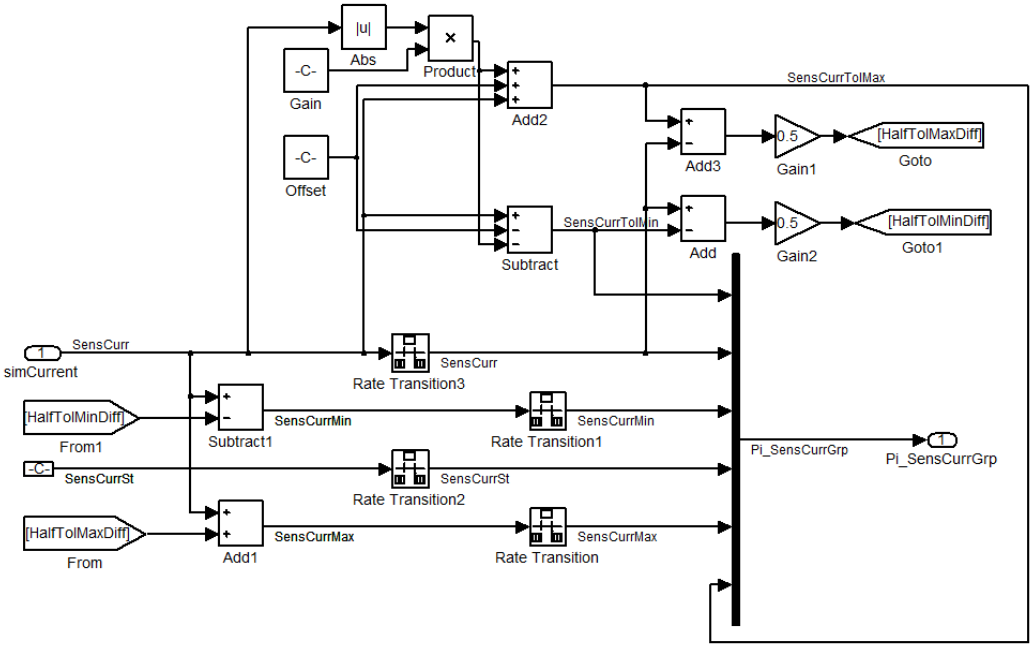

Abbildung 2.4: Signalgruppe der Stromsignale. Zu erkennen ist die Generierung der `Min/Max`- und der `TolMin/TolMax`-Werte des Stromsignals

Aufgrund der in (siehe Abschnitt 2.3.1.3) beschriebenen Problematik bei der Generierung des Durchschnittswerts der Zellspannung, wird in dieser Implementierung der Ansatz der Rückskalierung der Systemspannung für die Zellspannung verwendet. Aus Erfahrungswerten geht aus dieser Variante hervor, dass bei den Extremwerten des Ladezustandes somit genauere Zellspannungswerte generiert werden.
Ebenso ist eine Absicherung der maximalen Zellspannungen in diesem Projekt auf einen Maximalwert von $3,8V$ implementiert, da in den Prüfstandsdaten, eine durch Aufzeichnungsfehler bedingte, für die Simulation unplausible, Überschreitung dieses Werts nicht auszuschließen ist.
Alle Signale an den BMS-Eingängen, die auf Prüfstandsdaten basieren, haben eine Anpassung an die globale Abtastzeit von $20ms$ gemeinsam (siehe Abschnitt 2.2.6).

2.2.2.1 Initialisierung

Im Start-Skript `Simulation_FRAME_ini` wurde zu Beginn der Aufruf des Auswahldialogs `selectionDialog` implementiert, der im Verzeichnis /Data alle darin gespeicherten Testzyklen auflistet und dessen Dateiname als aktuellen `cycleName` der Simulation definiert. Es sollte sich dabei um einen der in (siehe Abschnitt 2.1.5.1) generierten Fahr- oder Testzyklen handeln, um eine Simulation initiieren zu können, da in dieser Darstellung alle `.mat`-Files im Ordner angezeigt werden.
Der weitere Initialisierungsvorgang kopiert die der BMS-Version entsprechenden `.mex`-Files für die Ausführung der S-Functions, sowie die `parameter_const_list` (für den Fall, dass die Simulink®-Module verwendet werden) aus dem Quellverzeichnis der aktuellen BMS-Simulationsmodelle (siehe Abschnitt 2.2.1.1). Für die in (siehe Abschnitt 2.1.6) beschriebene Methode, Parameter aus dem Workspace benutzen und manipulieren zu können, wird dafür die gleichnamige Struktur mit dem Codeabschnitt aus Listing 2.1 angelegt.

```
1 evalin('base', 'clear parameter_const_list__');
  evalin('base', 'parameter_const_list__ = load(''parameter_const_list__''
      ');');
```

Listing 2.1: Das Laden der `parameter_const_list` in den Workspace

Je nach Signalquelle (siehe Abschnitt 2.1.5.1) soll in weiterer Folge die Initialisierung angepasst werden. Stammen die Signale vom Prüfstand, muss lediglich deren Umfang auf Vollständigkeit geprüft werden (siehe Abschnitt 2.3.2.5). Die Startwerte für SOC, Temperatur und die Hysterese werden aus den Anfangswerten der jeweiligen Prüfstandsignale ermittelt und (wenn vorhanden) angepasst.

Sollte der Testzyklus aus einem Skript nach Methode 2 aus Abschnitt 2.1.5.1 generiert worden sein, wurden die entsprechenden Startwerte bereits definiert und müssen bei der Initialisierung lediglich als solche erkannt und geladen werden. Die einfachste Variante, den Typ des Fahrzyklus zu überprüfen, stellt die Abfrage auf eine vorhandene Prüfstands-Datenstruktur dar. Sollte sie nicht vorhanden sein, kann von einem per Skript generierten Zyklus ausgegangen werden.
Die übrigen NVM-Werte werden mit Erfahrungswerten für ein allgemeingültiges, akzeptables Startverhalten der betroffenen Funktionalitäten bedatet, können aber bei Bedarf an dieser Stelle im Skript editiert werden. Die Initialisierung der ZM relevanten Struktur wird per Skriptaufruf eingeleitet. Bis auf die in (siehe Abschnitt 2.2.1.2) vorgenommenen Manipulationen läuft der Vorgang in einem unabhängigen Rahmen, im Skript `StartCellModel_basic_CI`, ab.
Der Hinweis auf eine mögliche Überschreitung des Arbeitsspeichers soll als grobe Abschätzung des Simulationsausgangs betrachtet werden. Der Grenzwert von $7000s$ (siehe Abschnitt 2.3.2.2) bei 2 GB Arbeitsspeicher stammt aus Erfahrungswerten, die aus den Simulationen bei der Funktionsanalyse (siehe Abschnitt 2.4) der vorliegenden Arbeit stammt. Sie wird mit Hilfe der MATLAB® Funktion `warndlg` erstellt. Konnte der Initialisierungsvorgang erfolgreich abgeschlossen werden, wird am Ende per MATLAB® `msgbox` in `createInfoDialogBox` eine Zustandsinformation angezeigt, die Auskunft über die Startwerte von SOC, Hysterese, Temperatur, der Start- und Endzeit des Testzyklus, des Temperatursimulations-Modus aus dem ZM und der Initialisierungsquelle der Start-LUTs gibt.

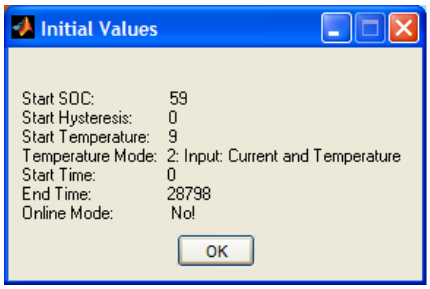

Abbildung 2.5: Simulationshinweise im `InfoDialog`

Als alternative Variante der Initialisierung von Start- und NVM-Parameter, kann ein Endzustand eines bereits simulierten Testzyklus herangezogen werden (siehe Abschnitt 2.1.4.5). Der Aufruf davon geschieht automatisch im Rahmen der Simulation längerer Fahrzyklen (siehe Abschnitt 2.1.5.2) oder manuell mit Hilfe der entsprechenden Schaltfläche im Simulation_FRAME (siehe Abschnitt 2.1.4.2). Dabei wird das

Skript `loadLastCycleState` gestartet. Es erleichtert die Auswahl des vom Benutzer gewünschten, vorangegangen Fahrzyklus aus dem Ordner /Results und lädt dessen Endzustandsdaten in den Workspace. Dabei werden die aktuellen Zustandsdaten überschrieben.

2.2.2.2 NVM

Die Implementierung einer NVM-Nachbildung benötigt die bei der Initialisierung angelegte Datenstruktur im Workspace, um die Simulation per `from_Workspace`-Blöcke die Startwerte auslesen zu können. Die Abspeicherung im Workspace nach Simulationsende erfolgt per `to_Workspace`-Block für jedes NVM-Signal extra, indem der letzte eingehende Wert der Simulation gespeichert wird.
Die Alternative zur NVM-Bedatung erfolgt über die `Konstant`-Blöcke und die entsprechende `Switch`-Stellung davor. Diese Möglichkeit soll eine zusätzliche Methode für das Setzen eines Startwerts darstellen, ohne die NVM-Werte im Workspace manipulieren zu müssen, was bei der Verwendung von Prüfstandsdaten einen eher aufwändigen Schritt darstellt.

2.2.3 Modelloptimierungen

2.2.3.1 Tausch der Mdls

Die Problematik aus (siehe Abschnitt 2.1.7.1) verlangt nach einer Möglichkeit, die Module im BMS ohne großen Aufwand zwischen S-Functions und Simulink®-Modulen austauschen zu können. Deshalb wurde eine überschaubare Auflistung mittels simpler graphischer Oberfläche auf Basis des *Graphical User Interface* (GUI) aus (siehe Abschnitt 2.2.6) implementiert, die per Kontroll-Box die Auswahl für den entsprechenden Modul-Austausch ermöglicht. Die Richtung des Tauschvorhabens wird mit Hilfe der Modul-Quellen-Bezeichnung bei der Übergabe der Funktionsparameter bestimmt (siehe Listing 2.2).

```
replaceMdlsInBMS('init','Simulation_FRAME','Ctap_BattMdlQm_SL')
%oder
replaceMdlsInBMS('init','Simulation_FRAME','Ctap_BattMdlQm_SFUN')
```

Listing 2.2: Der Aufruf der Funktion `replaceMdlsInBMS` für beide Tauschvorhaben

Bei detektiertem Selektieren der Kontroll-Box wird, in Abhängigkeit der Richtung des Modultausches, die Funktion `replaceBlock` (siehe Abschnitt 2.2.6) mit den entsprechenden Übergabe-Parametern aufgerufen, um die Aktion durchzuführen. Für die Funktionalität ist es aber notwendig, dass die betroffenen Modelle vorher per `load_system` in den Arbeitsspeicher geladen wurden. Beim Abschließen des Tauschvorganges wird die Funktion `replaceLUTwithSimplified` aufgerufen, die am Ende, unabhängig vom vergangenen Arbeitsschritt, die vordefinierten LUTs gegen die vorbereiteten ersetzt, wie im Anschluss beschrieben. Dieser Schritt wird jedoch nur ausgeführt, wenn S-Functions gegen Simulink®-Module getauscht werden, da der umgekehrte Schritt aufgrund der Unabhängigkeit der S-Functions von den LUTs keinen Sinn ergeben würde.

2.2.3.2 Automatisierter Austausch der LUTs

LUT-Bibliothek: Die in Abschnitt 2.1.6 beschriebene Problematik der BMS-LUTs, in den Bereichen Laufzeit und Bedatungsmöglichkeiten als umständlich zu gelten, wird mit dem Austausch gegen die Standard-Simulink®-LUTs behoben.
Grundlage dafür bildet die generierte Bibliothek in Form eines Simulink®-Mdls (`Lut-Bib`), in der sich alle zum Austausch vorbereiteten LUTs befinden. Es werden vom folgenden Algorithmus also nur jene LUTs zum Austausch in Betracht gezogen, die darin vorkommen. Ein wichtiger Punkt für die korrekte Einbindung ins BMS ist der, dass die übergeordneten Subsysteme, in denen sich diese LUTs befinden, Form und Interface-Struktur nach außen hin beibehalten.
Der große Nachteil der im BMS integrierten 3D-LUTs wurde in (siehe Abschnitt 2.1.7.2) beschrieben. Die Lösung dieses Problems bietet der Austausch gegen die von Simulink® bereitgestellten 3D-LUTs. Die notwendige Bedatung erfolgt direkt aus der `parameter_const_list` (siehe Abschnitt 2.1.6), mit der Einstellung für lineare Interpolation zwischen den Stützstellen bzw. ohne Extrapolation an den Grenzen der Wertebereiche. An der zu tauschenden Stelle bleiben im betroffenen Subsystem, neben der 3D-LUT, die Sättigungslimit und die noch nicht in Verwendung befindliche EOL-Berücksichtigung (LUTs siehe Anhang A.2).
Für die ein- und zweidimensionalen LUTs betrifft die Manipulation nur die Bedatung mittels Methode 2 (siehe Abschnitt 2.1.6) zur Laufzeitbeschleunigung und der vereinheitlichten Parametrierung.

LUT-Austausch Skript: Das MATLAB®-Skript zum Austausch der LUTs `replaceLUTwithSimplified` generiert nach Aufruf eine Liste mit allen LUTs aus der `Lut-Bib`,

die somit in weiterer Folge im Modell berücksichtigt werden. Für jede LUT aus dieser Liste wird die Funktion `replaceBlock` (siehe Abschnitt 2.2.6) aufgerufen, die im definierten Modell mit dem Pendant aus der Bibliothek getauscht wird.

Aufruf des LUT-Tausch-Skripts: Der Aufruf der beschriebenen Funktionalität erfolgt am Ende der Funktion `replaceLUTwithSimplified`, welche die BMS S-Functions gegen ihre Simulink®-Module tauscht (siehe Abschnitt 2.1.7.1). Da bei diesem Arbeitsschritt der Zeitaufwand vernachlässigbar ist, wird nach jedem beliebigen Tauschvorhaben der gesamte Algorithmus auf alle LUTs angewendet.
Alternativ kann die Funktion manuell in MATLAB® aufgerufen werden. Dabei ist aber auf die Syntax des Funktionsaufrufes zu achten.

2.2.4 Simulationsstart/Simulation längerer Zyklen

Nach der Initialisierung kann über die Simulink®-Schaltfläche `Run` die Simulation gestartet werden. Sollte der zu simulierende Fahrzyklus aufgrund seiner Laufzeit, die Kapazität des Arbeitsspeichers vom Entwicklungsrechner überschreiten, kann mit dem Skript `simulateExtensiveDataPackage` eine Aufteilung der Simulationsdauer vorgenommen werden.
Am Beginn des Algorithmus müssen bereits geladene Fahrzyklendaten verworfen werden. Sonst würde bei der Abtastzeitanpassung der Eingangssignale per `interpolate_input_20ms` ein Fehler bei der Zuweisung zu einer bereits vorhandenen, ev. längeren Signalmatrix entstehen, da die generierten Spaltenvektoren nacheinander in die 2-Spalten-Matrix geladen werden (erste Spalte Zeit-, zweite Spalte Signalvektor, siehe Listing 2.3).

```
1 [Current(:,1) Current(:,2)] = interpolate_input_20ms(data.CAN.
    Bat_ElecStat.Time, data.CAN.Bat_ElecStat.Bat_Curr);
```

Listing 2.3: Generierung der 2-Spalten Strom-Matrix aus dem (vereinfacht dargestellten) Prüstandsdatensatz

Die Abfrage über die Anzahl der gewünschten Strukturteile erfolgt im Dialog `inputdlg`. In weiterer Folge markiert sich der Algorithmus den ersten Teil der Zyklen-Serie mit einem Flag, bevor er die einzelnen Fahrzyklusteile, in einer `for`-Schleife, der Reihe nach simuliert. Nur den ersten initialisiert er mit den Startparametern, bei allen weiteren wird der zuvor gespeicherte Endzustand (siehe Abschnitt 2.2.4.1) vor der Simulation wieder geladen. Die Anzahl der Zyklusteile definiert auch die

Anzahl der Schleifendurchläufe. In jedem Durchlauf der Schleife wird der aktuelle Fahrzyklus mit den Informationen des aktuellen Schleifendurchganges mittels `splitInputStruct` geteilt. Die Nummer der Schleifeniteration definiert aber, welcher dieser Teile nun der aktuelle Teil-Fahrzyklus für die Simulation ist.
Dieser Teil-Fahrzyklus dient dann als Initialisierungsquelle für den anschließenden Simulationsdurchgang. Dafür muss jedoch noch die Start- und die Endzeit angepasst, bzw. der zeitliche Offset am Ende gespeichert werden. Die Speicherung der einzelnen Ergebnisse erfolgt nach dem in Abschnitt 2.2.4.1 beschriebenen Schema, mit dem Unterschied, dass der Name des Fahrzyklus um den Schleifenindex erweitert wird. Der zwecks Automatisierung aktivierte Callback am Ende der Funktion (siehe Abschnitt 2.2.4.1) muss für den Aufruf dieses Algorithmus vor der Simulation deaktiviert werden, um die Problematik aus Abschnitt 2.3.2.7 zu umgehen.

2.2.4.1 Speicheralgorithmus

Simulation_FRAME: Am Ende eines Simulationsvorganges wird mit Hilfe des globalen Modell-Callbacks[10] das Skript `saveScopeDataAndEndState` aufgerufen, um die im Workspace aufgezeichneten Strukturen der implementierten Scopes zu sichern. Diese Variante der Datenspeicherung wurde aufgrund der Praxistauglichkeit gewählt, da per Scope bereits während der Laufzeit der Simulationsverlauf verfolgt und gleichzeitig gespeichert werden kann. Die darin per `save_Data_to_Workspace` aktivierte Funktion, die aufgezeichneten Daten im Workspace abzulegen, hat zum Einen den Vorteil, die Scope-Ansicht mit dem MATLAB®-Befehl `simplot(name)`[11] sofort angezeigt werden kann, zum Anderen bietet sie die Möglichkeit, die aufgezeichneten Signale aus deren zweidimensionaler Struktur herauszufiltern und dank vorhandener Zeitinformation für weitere Darstellungsformen zu verwenden.

VCM_Mdl: Dieses Modell speichert die generierten Simulationsergebnisse ebenfalls mit Hilfe eines Scopes, welches die Signale für die Archivierung auf der Festplatte vorbereitet. Nach Abschluss einer Simulation im VCM_Mdl wird die optionale Speicherung des Ergebnisses per `questdlg` abgefragt. Anschließend muss mittels `inputdlg` der Ergebnisname gewählt werden, wobei die Endung „_reduced“ vorgegeben wird, um ein Überschreiben eines vorhandenen Ereignisses zu verhindern. Das Ergebnis wird ebenfalls im /Result-Ordner abgelegt, worauf automatisierte Skripte

[10] Damit sind die Callbacks aus den `Model-Properties` gemeint
[11] `name` muss jener der im Scope gespeicherten Struktur sein

der Signaldarstellung zurückgreifen können.

Zustandsspeicherung: Um am Simulationsbeginn eines Fahrzyklus auf seine Belastungsgeschichte schließen zu können, findet im Rahmen der Ergebnisspeicherung die Zustandsspeicherung statt. Dabei werden die benötigten NVM-Werte, deren letzte Werte mittels `to_Workspace`- Simulink®-Block gesichert werden, ebenfalls gespeichert. Dieser Speichervorgang findet getrennt statt, da die simple Zustandsspeicherung für Simulationsvorgänge eventuell getrennt benötigt werden könnte.

Abschnitt 2.2.1.1 erwähnt, dass bei erstmaliger Ausführung des Skripts, der Ordner /Result im aktuellen Arbeitsverzeichnis angelegt wird, um darin die Ergebnisse aller simulierten Fahrzyklen abzuspeichern. Pro Fahrzyklus wird dafür ein Ordner mit dessen `cycleName` angelegt, in dem die per Scopes aufgezeichneten Datenpakete zu finden sind. Sie müssen vorher per `save`-Funktion in ein `.mat`-File gespeichert werden, welches, in ausbalancierter Form, die Scope-Strukturen beinhalten, aber auch den gespeicherten Endzustand. Dabei ist zu beachten, dass die entstehende Speichermenge die maximal zulässige Dateigröße nicht überschreitet. Am Ende findet die Zustandsspeicherung statt.

2.2.4.2 Signaldarstellung der Ergebnisse

Auf einem Standard-Entwicklungsrechner (siehe Abschnitt 2.1.1) ist es aus Speichergründen notwendig, die darzustellenden Simulationssignale mengenmäßig und zeitlich zu begrenzen, um sie archivieren und bei Bedarf aufrufen und präsentieren zu können. Deswegen richtet sich diese Funktion an die Darstellung von oft betrachteten, für die Funktionsanalyse (siehe Abschnitt 2.4) aussagekräftigen Signale, die für weitere Anwendungsfälle bei Simulation des Simulaton_FRAMEs oder des VCM_Mdls, adaptiert werden kann.

Simulaton_FRAME: Die Funktion `show_bms_test_cycle_output` wird manuell im Editor oder per Schaltfläche in der Simulationsumgebung gestartet. Darin sucht die Funktion `searchCLMat` im Ordner /Results nach allen darin befindlichen `.mat`-Files, welche Simulationsergebnisse archivieren. Per `selectionDialog` wird das gewünschte File ausgewählt und geladen. Für den Signalvergleich mit Referenzdaten wird im Ordner /Data mit `searchCLMat` nach Fahrzyklen gesucht. Fällt die Suche

erfolgreich aus, wird ein entsprechendes Flag gesetzt, um den Vergleich in weiterer Folge berücksichtigen zu können. Die Auslösung der einzelnen Signale aus der vom Scope generierten Datenstruktur, wird mit der Hilfsfunktion `structfind` erreicht. Sie sucht in diesem Fall nach dem Namen eines in der Struktur befindlichen Signals und gibt dessen Index in der zweidimensionalen Struktur zurück. Voraussetzung dafür ist, dass die aufgezeichneten Signale im Simulink®-Modell zuvor benannt wurden. Bei der Signaldarstellung mittels `plot` muss für den Fall, dass die Simulation nicht bei Sekunde Null beginnt, der zeitliche Offset geladen und dazu gerechnet werden.

VCM_Mdl: Der Signal-Monitor des Simulation_FRAMEs, der aus einer Reihe von Scopes besteht, bereitet die Eingangssignale des VCM_Mdls für die Signaldarstellung vor und stellt ebenfalls dessen Referenz-Ausgangssignale bereit.
Implementiert wurde die Darstellungsroutine im Skript `comparingVcmOutput`, in der, nach getroffener Auswahl des Simulationsergebnisses, die Ergebnisse mit Hilfe der Funktion `plot` dargestellt werden.

2.2.5 Implementierung der Parameterreduktion

Die Durchführung der Parameter-Reduktion wird unabhängig davon, ob sie im Simulation_FRAME oder im VCM_Mdl benötigt wird, in beiden Simulationsumgebungen per Skript oder per Schaltfläche gestartet. Die Auswahl der gewünschten Stützstellen wird durch ein implementiertes GUI erleichtert.
Manipuliert werden die Stützstellen und die Achseninformationen der ein-, zwei und der dreidimensionalen LUTs für BOL, die sich in den, während der Initialisierung geladenen, BMS-Simulink®-Modulen befinden. Die Implementierung des Algorithmus findet in `replaceLUTwithReduced` statt.

Vorbereitung: Die Parameterreduktion manipuliert das in der Ordnerstruktur liegende `.mat`-File `parameter_const_list`, damit alle in Abschnitt 2.1.6 beschriebenen Parametrier-Methoden von den Änderungen betroffen werden. Um von vorangegangenen Parameterreduktionen unabhängig zu sein, wird sie vorher mit der unveränderten `parameter_const_list_original` überschrieben (siehe Abschnitt 2.2.1.1). Im Anschluss wird die neue Parameter-Liste in den Workspace geladen (siehe Listing 2.1). Ebenso muss für weitere Arbeitsschritte der originale Messdatensatz der SOC/OCV-Stützstellen, in den Workspace geladen werden. Dieser Messdatensatz in Form eines `.mat`-Files beinhaltet Ergebnisse der Zellcharakterisierung.

Struktur: Die Stützstellen der Parametervektoren aus der `parameter_const_list` werden in eine neue Struktur kopiert, um sie der Funktion `selectReducingValues` (siehe letzter Teil im Abschnitt 2.2.5) übergeben zu können. Dabei ist auf die Reihenfolge der Daten zu achten, da sie zur Beschriftung des implementierten GUI dienen. Die Information über die ausgewählten Stützstellen wird in Form ihrer Vektor-Indizes in `reducingInfo` zurückgegeben. Mit diesen Indizes werden aus der `parameter_const_list` die übrigen Stützstellen gelesen und in eine temporäre Struktur gespeichert. In allen Parameterstrukturen muss dabei mehr als eine Stützstelle nach der Auswahl vorhanden sein, um die LUTs in Simulink® bedaten zu können. Für den Fall, dass nur eine Stützstelle ausgewählt wurde, wird sie als zweite Stützstelle dupliziert, um Simulink® die Extrapolation zu ermöglichen.

Relevante Stützstellen: Die zuvor in den Workspace geladenen Messdaten werden mit der neu generierten, temporären Struktur überschrieben, um im Anschluss einen zur Verfügung gestellten Algorithmus zur Datenoptimierung zu starten. Die dafür relevanten Stützstellen werden im GUI per Checkbox markiert. Daraus generiert der Algorithmus eine neue, zweidimensionale LUT, die nur die ausgewählten Stützstellen abbildet. Dabei wird jeweils der SOC oder der OCV über die ausgewählten Temperaturbereiche mit OCV- im ersten, oder SOC-Parametern im zweiten Fall gespeichert.
Kann der Algorithmus erfolgreich ausgeführt werden, ist letztendlich an dessen Speicherort eine zellspezifische Datenstruktur zu finden, deren Parameter dem BMS für die jeweilige Zelltechnologie als Referenz dienen.
Der zu parametrierende Temperaturbereich kann über eine obere und eine untere Grenze global eingeschränkt werden. Dafür wird auf die Funktion `reduceUnnecessaryTemp` zurückgegriffen, der die Temperaturgrenzen aus `reducingInfo` als Übergabeparameter dienen. Die Funktion lässt nur jene Stützstellen in den neuen, temporären LUTs

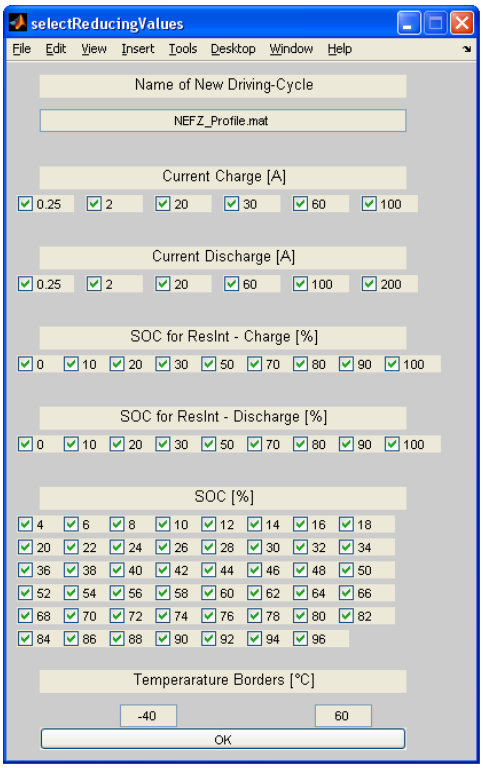

Abbildung 2.6: Simples GUI zur Auswahl der relevanten Parameterstützstellen

zu, die innerhalb der Grenzen liegen. Dabei ist zu beachten, dass Funktionalitäten im BMS unterschiedlich viele Stützstellen der Temperatur ausweisen, wobei an deren Grenzwerten eine lineare Extrapolation stattfindet. Wird die Reduktion ausgeführt, werden alle LUTs im BMS an die neuen Temperaturgrenzen angepasst. Die Funktion des BMS setzt eine Stützstelle bei -20°C voraus, welche aber im Messdatensatz der SOC/OCV Stützstellen nicht vorhanden ist. Bei der Softwaregenerierung wird deshalb die niedrigste Temperaturstützstelle für diesen Tieftemperaturbereich kopiert. Bei der Reduktion muss deswegen zusätzlich abgefragt werden, ob diese Stützstelle überhaupt benötigt wird, um die Parameter manuell zu kopieren. Ansonsten werden nur die ohnehin vorhandenen Stützstellen beachtet.
Die Stützstellen des Belastungsstroms werden in Lade- und Entladerichtung getrennt betrachtet, um eine einseitige Vernachlässigung der Belastungsanforderung, z.B. um einen einseitigen Teildurchlauf der Resistance-Tests (siehe Abschnitt 1.5.3.10), definieren zu können. Sie betrifft jene Einträge in der `parameter_const_list`, aus denen die Werte der Innen- und der Polarisationswiderstände über LUTs abgebildet werden.
Aufgrund einer stark eingeschränkten Skalierung jenes SOCs, von dem die Innenwiderstandsabbildung abhängt, werden seine Stützstellen eigenständig in Lade- und in Entladerichtung per Checkbox manipuliert.

Abschluss: Am Ende der Ausführung der Parameterreduktion wird das `.mat`-File der `parameter_const_list` im Arbeitsverzeichnis aktualisiert und anschließend in seiner manipulierter Form in den Workspace geladen. Die darin enthaltenen, nun manipulierten LUTs, werden direkt vom BMS referenziert.

`selectReducingValues`: Die Funktion baut auf der Basis des GUI aus Abschnitt 2.2.6 eine graphische Auswahlmaske auf, die mit Hilfe der übergebenen Struktur zur Laufzeit beschriftet wird (siehe Abb. 2.6). Die Funktion basiert auf der Grundlage, dass in Abhängigkeit der Länge der übergebenen Parametervektoren dazugehörige Index-Vektoren gebildet werden. Je nach betätigter Checkbox des entsprechenden Datensatzes wird der referenzierte Index entfernt oder hinzugefügt.
Die Inhalte der Textfelder werden hingegen direkt editiert. Für beide Eingabemöglichkeiten wird am Ende eine einzige Ergebnisstruktur erstellt, in der die neue Struktur an das aufrufende Skript zurückgegeben wird. Zusätzlich wird in der Ergebnisstruktur die Tatsache gespeichert, ob der Vorgang abgebrochen wurde oder nicht, um im aufrufenden Skript den weiteren Berechnungsverlauf aufhalten zu können.

2.2.6 Hilfsfunktionen für die Implementierung

`selectionDialog`: Dieser Funktion werden ein Ordnerpfad und ein Titel als Parameter übergeben. Sie sucht im Ordner nach allen Dateien, die sich darin befinden, listet die daraus gefundenen `.mat`-Files in einer Dialogbox auf und gibt den Namen der selektierten Datei zurück. Sollte es sich nicht um einen gültigen Fahrzyklus handeln, der den Kriterien entspricht (siehe Abschnitt 2.1.5), wird der Vorgang abgebrochen (siehe Listing A.1).

`interpolate_input_20ms`: Die bereits erwähnten Probleme der abweichenden Abtastzeit von $20ms$ zwischen den verschiedenen Signalen eines Fahrzyklus können mit dieser Funktion behoben werden. Aus den als Parameter übergebenen Zeit- und Datenvektoren mit beliebiger Abtastzeit, werden mit Hilfe der MATLAB®-Funktion `interp1` ein Zeit-und Datenvektor mit $20ms$ Abtastzeit generiert (siehe Listing A.2).

`replaceBlock`: Die des Öfteren geforderte Auswechslung von identen Simulink® Subsystemen mit unterschiedlicher interner Funktionalität oder Komplexität, kann mit der Funktion `replaceBlock` bewerkstelligt werden. Sie basiert auf der MATLAB®-Funktion `find_system`, die nach Subsystemen oder Blöcken in Simulink®-Models sucht und die Ergebnisse, also auch bei mehreren Treffern, als Referenzenliste zurückgibt. Voraussetzung ist, dass die Namen des zu durchsuchenden Modells und der zu suchende Komponente als Übergabeparameter angegeben werden. Diese Identifizierung geschieht für die vorhandenen und die zu ersetzenden Bauteile auf dieselbe Weise. Per `get_param` und `getfullname` können die Eigenschaften der gefundenen Blöcke geladen werden, um sie per `add_block` und `set_param` gegen ihre Pendants austauschen und einrichten zu können (siehe Listing A.3).

GUI: Für die vereinfachte Anwendung von Auswahlmöglichkeiten wurde der Ansatz einer einfachen graphischen Implementierungsmöglichkeit per MATLAB®-GUI herangezogen und als Grundlage benutzt (vergleiche [Mat12] im Anhang, Listing A.6).
Der Ansatz beruht darauf, dass bei einer Aktion im GUI, die Basisfunktion immer wieder mit der entsprechenden Aktion als Parameter aufgerufen wird. Die unterschiedlichen Aktionen werden per `switch/case`-Entscheidungsbaum unterschieden. MATLAB® stellt mit `uicontrol` ein simples Werkzeug zur Verfügung, welches Art,

Position, Beschriftung und Anweisung eines graphischen Elements definiert. Die Anweisungen werden in diesem Fall als lokale Callbacks definiert, die mit Hilfe des per `mfilename` zugewiesenen Funktionsnamen, die Basis-Funktion erneut aufruft, um anhand das `case`-Statements des Callback-Parameters die zugewiesene Funktionalität auszuführen.
Die lokale Datenstruktur wird dabei eng an die erstellte Figure verknüpft, um jedem `case`-Statement einen Aufruf und eine Erweiterung von GUI und Daten zu ermöglichen. Beim Erstellen der Figure wird auch die Terminierungs-Option definiert, die unabhängig vom Funktionsende, nach beabsichtigter Aktion, einen Terminierungs-Algorithmus starten kann.

`splitInputStruct:` Mit dieser Funktion wird das Vorhaben vereinfacht, eine umfangreiche MATLAB®-Struktur, beispielsweise ein Fahrzyklus vom Prüfstand, in mehrere, kürzere Strukturen zu zerlegen. Die Funktion arbeitet sich rekursiv durch den Datenbaum der übergebenen Struktur, um den letzten Datensatz am Ende zu teilen. Der Divisor, in wie viele Teile geteilt werden muss, bzw. der Index, bei welchen Teil sich der Algorithmus gerade befindet, wird per Übergabeparameter von der aufrufenden Funktion vorgegeben. Am Ende gibt die Funktion, neben dem kleinen Teilstück des Fahrzyklus, die Grenzindizes der Datenvektoren zurück (siehe Listing A.4).

2.3 Aufgetretene Probleme beim Entwurf und bei der Umsetzung

2.3.1 Probleme im Entwurf

2.3.1.1 Unveränderliche Signal-Abtastzeit des Systems

Die globale Abtastzeit des BMS wurde von Beginn der Steuerungssoftware-Entwicklung an mit $20ms$ definiert und nach diesen Vorgaben implementiert. Dieser Schritt beeinflusst die weitere Benutzung der Software, vor allem in Verbindung mit den S-Function-Modulen, da sie keine Option zur Veränderung dieses Parameters zur Verfügung stellen. Aus diesem Grund muss das ZM sowie die Eingangssignale des BMS auf die $20ms$-Abtastzeit kalibriert werden, um die Funktion des Systems sicher zu stellen.

2.3.1.2 Nachbildung des Grundrauschens des Stromsensors

Das am Realsystem laufende BMS erwartet im Betrieb, wenn keine Belastung am Stromsensor anliegt, das Grundrauschen des Stromsensors als Stromeingangssignal. Konkret muss immer ein gültiger, plausibler Toleranzbereich größer null in der Stromsensor-Signalgruppe generiert werden (siehe Abschnitt 2.2.2). Sollte der Toleranzbereich null werden, ist keine korrekte Simulation möglich. Im praktischen Fall wird deshalb in Ruhephasen eine Stromamplitude von mindestens $0,5mA$ an den `SensCurrTolMin/Max` angelegt, unter Belastung $0,5\%$ vom aktuellen Strom (siehe Abschnitt 2.2.2). Dieser Wert ist dem realen Stromsensor-Offset nachempfunden. Grund dafür ist die Fehlerberechnung der Spannungsabfälle an den Innen- und Polarisationswiderständen, welche diese Toleranzwerte (`Min/Max`) als Divisor erhalten.

2.3.1.3 Berechnung der durchschnittlichen Zellspannung

Für den Entwurf der Signalgenerierung war die Entscheidung notwendig, nach welchem Ansatz die Durchschnittsspannung über den gesamten Zellverband gebildet werden soll. Generell gilt die Signalgenerierung aus dem Mittelwert zwischen der minimalen und der maximalen Zellspannung als guter Ansatz. Sollten diese beiden Signale in der Simulation vom ZM generiert werden, sind aber, in Zellspannungsbereichen nahe den Spannungsgrenzen, Berechnungsungenauigkeiten zu erwarten. Grund dafür ist die ehemalige Simulationsanforderung des ZMs auf ein kleineres

SOC-Fenster[12], weshalb es bei veränderten Ladezustandsgrenzen Schwächen in der Rechengenauigkeit zeigt. Deswegen wird die Durchschnitts-Zellspannung aus der skalierten Systemspannung generiert.

2.3.2 Probleme während der Umsetzung

2.3.2.1 Start-Zeitpunkt muss bei null liegen

Aufgrund von Rückkoppelschleifen und Initialisierungsvorgängen existiert im Input-Block eine Sprungfunktion, die erst nach einigen Abtastzeitschritten den NVM und somit die Systemfunktionalität freigibt. Die Zeitspanne bis zur Freigabe entspricht der Startzeit der Software. Aus praktischen Gründen startet die Zeitrechnung beim Wert null, weshalb keine korrekte Funktion in Gang gesetzt werden kann, wenn die `StartTime` von null abweicht. Das ist vor allem bei der Aufteilung eines Testzyklus in mehrere Pakete zu beachten, wenn bei einer beliebigen Zwischenzeit gestartet werden soll.

2.3.2.2 Abschätzung des Simulationsspeicherbedarfes

Im Falle einer Simulation eines längeren Fahrzyklus auf dem Standard-Entwicklungsrechner (siehe Abschnitt 1.5.1), muss beachtet werden, dass die dabei entstehenden Simulationssignale in ihrer Datenmenge die Kapazität des Arbeitsspeichers überschreiten. Ein „Out of Memory“-Error wäre die Folge. Beeinflusst wird die mögliche Simulationsdauer von der Anzahl der Simulink®-Module während der Simulation, der Anzahl der Scopes und der geladenen Datenmenge im Workspace. Für die in Abschnitt 2.4 durchgeführten Simulationen war eine Zyklendauer von bis zu 7000 Sekunden möglich.

2.3.2.3 Absicherung Path zum Arbeitsverzeichnis

Sollte der Simulation_FRAME aus dem Windows®-Explorer heraus geöffnet werden, ist es für den Simulink®-Modelbrowser nicht möglich, das korrekte Arbeitsverzeichnis der Simulationsdateien von sich aus zu identifizieren. Werden in weiterer Folge Funktionen mit Hilfe der generierten Schaltflächen gestartet, kann nicht ausgeschlossen werden, dass eine andere, gleichnamige Datei, aus einem anderen Verzeichnis in der

[12] Mit SOC-Fenster ist die Definition von oberer und unterer Ladezustandsgrenze gemeint, in dem sich die Zelle im Rahmen ihrer Lebensdauer bewegt

eingestellten Liste der MATLAB®-Pfade, referenziert wird. Versuche, das Ordnerverzeichnis dynamisch an den Simulation_FRAME zu binden, blieben erfolglos.

2.3.2.4 SOC-Port Manipulation wegen Bug

In der verwendeten B-Muster Version des BMS ist ein Bug enthalten, der das Ausgangssignal `BatMinSocErrNeg` nicht über den Betrag von dessen NVM-Initialwert (`BatMinSocErrNegNvm`) steigen lässt. Der Grund dafür liegt im SOC-Modul, wo zwei Eingangsports im Initialisierungsblock vertauscht sind. Aufgrund der manuellen Korrektur im Simulink®-Modul des SOCs im Rahmen dieses Projekts konnte deswegen auf die äquivalente S-Function nicht mehr zurückgegriffen werden.

2.3.2.5 Unterschiedlicher Umfang des Inhalts der Prüfstandsdaten

Unterschiedliche Anforderungen bei Funktions-Tests am Prüfstand erfordern für jeden Fall spezielle Sammlungen von aufzuzeichnenden Signalen. In den meisten Fällen kann auf einen grundlegenden Datensatzumfang, bei bereits vorhandenen Prüfstandsdaten, zurückgegriffen werden, vor allem bei den Vehicle-CAN-Signalen. Bei den Development-CAN-Signalen muss jedoch bei der Implementierung automatisiert ablaufender Signaldarstellungen darauf geachtet werden, ein mögliches Fehlen eines Signals zur Laufzeit erkennen und mit Maßnahmen darauf reagieren zu können.

2.3.2.6 Fehler-Debuggen bei der GUI-Implementierung

Die in Abschnitt 2.2.6 benutzte, vereinfachte Generierungsmethode einer einfachen, graphischen Auswahl- und Eingabemaske, bringt den großen Nachteil mit sich, dass im Falle eines Programmierfehlers, der während der Laufzeit die Programmausführung unterbricht, MATLAB/Simulink® in einen unbenutzbaren Zustand versetzt.

2.3.2.7 Aufruf eines Callbacks während der Ausführung eines Callbacks

Zur Vereinfachung diverser Funktionsaufrufe bzw. zur Automatisierung im Ablauf des Simulationsalgorithmus wird auf Simulink®-Callbacks zurückgegriffen. Sie dienen dem Aufruf einer Funktion/eines Skripts, wenn ein Zustand im Simulationsablauf erreicht wird, z.B. das Öffnen eines Modells, das Starten einer Simulation oder das Erreichen des Simulationsendes. Nun kann es vorkommen (vergleiche Funktion in Abschnitt 2.2.4), dass eine Simulation, die mittels Callback gestartet wurde,

bei Erreichen eines bestimmten Zustandes einen weiteren Callback aufrufen soll. In der verwendeten MATLAB/Simulink®-Version 2010b wird diese Funktionalität nicht unterstützt. Deswegen muss ein verschachtelter Aufruf eines Callbacks verhindert, oder wie in Abschnitt 2.2.4 beschrieben, umgangen werden.

2.3.2.8 Fehlfunktionen in S-Functions

Der Bug aus Abschnitt 2.3.2.4 und weitere Fehlfunktionen, die während der Umsetzung entdeckt wurden, konnten in den Simulink®-Mdls für die Simulationen in Abschnitt 2.4.2 behoben werden. Da aus den Simulink®-Mdls aber die performanten S-Functions erstellt werden, sind darin die Fehlfunktionen weiterhin enthalten. Aus diesem Grund können die S-Functions mit diesen Fehlfunktionen nicht weiter verwendet werden. Auf den erhofften Geschwindigkeitsvorteil bei der Initialisierung der Simulation von bis zu 70%[13] muss deshalb in weiterer Folge verzichtet werden.

2.4 Funktionsanalyse

In diesem Abschnitt wird die Funktion der in dieser Arbeit entworfenen und umgesetzten Simulationsumgebung demonstriert, indem am Prüfstand gefahrene Testzyklen simuliert werden (siehe Abschnitt 1.7). Am klimatisierten Prüfstand (siehe Abschnitt 1.3.6) wurde eine PHEV-Vorserienbatterie aufgebaut, auf deren BMU das hier verwendete B-Muster BMS für die folgenden Tests geladen wurde. Aus Zeitgründen war die Anzahl der Test auf zwei begrenzt, wobei einer davon den Prüfstand sehr lange in Anspruch nahm. Zu erwarten ist eine Annäherung der Simulationsergebnisse, die vom modellbasierten BMS in der Simulationsumgebung generiert wurden, an jene vom Prüfstand, da die Signal- und Zustandsberechnungen auf derselben Softwareversion basieren.
Ebenso wird die erwartete Abweichung der Funktionalität dargestellt, die unweigerlich im Betrieb von reduzierten Parametersätzen in undefinierten Betriebsbereichen auftreten. Auch in diesem Fall, muss beim Signalvergleich die Softwareversion von Simulation und Referenz übereinstimmen.

[13] Der Wert stammt aus Simulationen, in denen die plausible Funktion der Mdls keine Rücksicht genommen wurde, um den Performancegewinn zu testen

2.4.1 Vorbereitung einer Simulation mit reduziertem Parametersatz

Im Simulation_FRAME bzw. im VCM_Mdl wird mit der Schaltfläche `Initialisation: Driving Cycles, Start Values` der gewünschte Fahrzyklus ausgewählt. Nach dem Ladevorgang gibt der `InfoDialog` Auskunft darüber, ob die Simulation über die Start-Funktion von Simulink®, oder über die Schaltfläche `Start the simulation of a long driving cycle` gestartet werden soll. Dafür sollte die verfügbare Speichermenge des verwendeten Entwicklungsrechners bekannt sein. Im zweiten Fall kann vor Simulationsstart mit der Wahl, in wie vielen Teilsimulationen der Test unterteilt werden soll, auch mit einer geringen Speichermenge im Entwicklungsrechner simuliert werden.
Beachtet werden muss vor dem Simulationsstart die Wahl des Temperaturberechnungsmodells im ZM. Dafür ist im Skript `SetCellModelData_CI` der Parameter `ModelParameters.TestCase` zuständig. Die Beschreibung dessen Bedatung ist an dieser Stelle nachzulesen. Im Fall einer Änderung muss die Simulation neu initialisiert werden.
Die Parameterreduktion wird über die Schaltfläche `Datenreduzierung` initialisiert. Zu Beginn sind alle Parameter markiert und die globalen Temperaturgrenzen des Systems, bzw. der Name des Tests, werden unverändert dargestellt (GUI in Abb. 2.6). Soll z.B. keine Parametrierung des Batteriesystem für Ströme über $60A$ stattfinden, werden die entsprechenden Stützstellen einfach abgewählt. Neue Temperaturgrenzen werden in die Eingabefelder eingetragen. Sollte ein Test mit unterschiedlichen Szenarien von reduzierten Betriebsbereichen simuliert werden, empfiehlt sich die Vergabe von jeweils treffenden Ergebnisnamen im obersten Eingabefeld.
Der Vorgang wird über die `OK`-Schaltfläche abgeschlossen, wobei damit die Neugenerierung der Softwareparameter gestartet wird, die einige Sekunden in Anspruch nimmt.
Nach dieser Anleitung wurden die folgenden Simulationen zur Demonstration der Funktionsweise der Simulationsumgebung durchgeführt. Trotzdem mussten zur Ergebnisdarstellung in diesem Abschnitt folgende Anpassungen vorgenommen werden:

- Im VCM wurde gegen Ende der Projektarbeit eine Fehlfunktion entdeckt, die den Index des `Hyst`-Werts vom Zellspannungssample nicht richtig berechnet. Dabei konnte beobachtet werden, dass der Betrag des Minimalwerts vom `SocVb` größer wurde, als der Betrag vom maximalen `SocVb`. Eine manuelle Generierung und Bedatung dieses Indexsignals im `Hyst`-Modul schaffte Abhilfe.

- Da in den demonstrierten Simulationen auch die zugrunde liegenden Parametersätze manipuliert werden, fällt die Wahl im Simulation_FRAME auf die Simulink®-Mdls. Theoretisch hätte an dieser Stelle ein Vergleich der Simulationsperformance zwischen S-Function- und Simulink®-Mdls stattgefunden. Dieser wurde aber aufgrund der bekannten Fehlfunktionen der verwendeten S-Functions nicht durchgeführt (siehe Abschnitt 2.3.2.8).
- Für die in diesem Abschnitt gezeigten Ergebnisse (Plots) wurden eigene MATLAB®-Skripts erstellt, die auf das Zusammensetzen der einzelnen Teilergebnisse für die Darstellung optimiert wurden.
- Der im Abschnitt 2.3.2.4 erwähnte Bug wurde durch Vertauschen der Port-Anbindungen von `SocMinErrNeg` und `SocMinErrNegNvm` manuell behoben.
- Für die Simulation von *OCV-Steps* und *SOC-Variation* wurde das ZM aus der Simulationsumgebung entfernt, um Systemressourcen zu sparen. Damit war eine verlängerte Simulationsdauer von bis zu $7000s$ möglich.

2.4.2 Die Testzyklen

2.4.2.1 Simulation-NEFZ

Zur praxisnahen Veranschaulichung eines Simulationsergebnisses wurde der NEFZ-Fahrzyklus [Wik13] gewählt. Dabei wurde keine Rücksicht auf diverse Einflussfaktoren wie unterschiedliche Leistungsverbraucher im Fahrzeug oder verändernde Wettereinflüsse genommen und eine Außentemperatur bei Simulationsstart von 20° C gewählt. Das Temperaturmodell im ZM übernimmt dabei die Berechnung des Betriebstemperaturverlaufs. Um den SOC-Bereich von $0 - 100\%$ komplett durchlaufen zu können, wurde der NEFZ drei mal in Folge simuliert. In Abb. 2.7 ist das Stromprofil des NEFZ zu sehen. Charakteristisch dafür sind hohe Strompulse am jeweiligen Abschluss eines Zyklus, verursacht durch die hohe Leistungsforderung [Wik13]. Ebenfalls sichtbar ist eine leichte Erwärmung des Batteriesystems während des Betriebes, da auf weitere Maßnahmen zur Kühlung des Systems keine Rücksicht genommen wurde.

2.4.2.2 Test OCV-Steps

Der Test *OCV-Steps* dient zur Analyse der SOC-Berechnung über den gesamten SOC-Bereich, während die Temperatur dabei über unterschiedliche OCV-Temperaturcharakteristiken schwankt (die OCV/SOC-Kennlinien sind in Abb. 1.11 von -20 bis

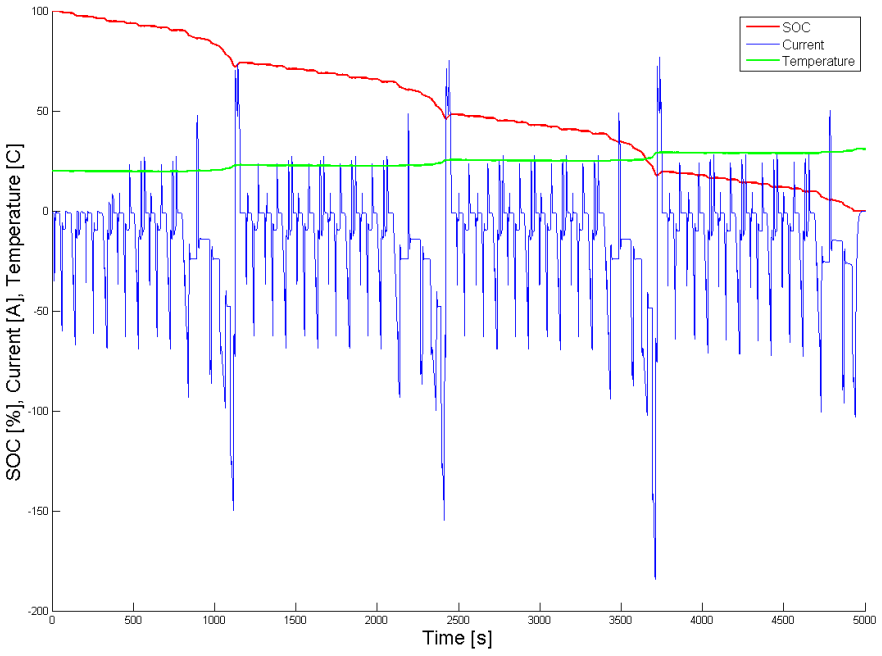

Abbildung 2.7: Das Ergebnis des NEFZ in Form von Strom, Temperatur und Ladezustandsberechnung

40° C dargestellt). Die Temperaturbereiche werden von den Stützstellen im OCV-Parametersatz definiert. Wichtig für den Test ist eine niedrige Stromamplitude der Lade-/ Entladeströme, um zum einen das Relaxationsverhalten einzudämmen und zum anderen eine SOC-Korrektur ermöglichen zu können.
Der Test *OCV-Steps* durchläuft den SOC in 10% Schritten, beginnend von SOC 0%, in Lade- und anschließend in Entladerichtung. Dafür wird am Beginn, zur Bestimmung der Kapazität, ein CapaDet-Test vollzogen und das System auf die Entladespannung von $2,1V$ eingestellt. Nach jedem Transfer von einem Zehntel der Kapazität wird anschließend das Relaxationsverhalten in einem Zeitfenster von 15 Minuten abgewartet, um am Ende davon die Batterietemperatur auf 10° C zu kühlen. Der Kühlvorgang wird aus Zeitgründen auf 30 Minuten begrenzt. Nach dieser Zeit wird der Lade-/ Entladevorgang fortgesetzt, mit der Bedingung, eine Betriebstemperatur von 30° C zu erreichen. Da die Eigenerwärmung des Systems dafür nicht ausreicht, muss auf die aktive Heizung vom Prüfstand zurückgegriffen werden. Erreicht das System die für Konstantströme eingestellte CV[14] von $3,51V$, wird der Ladestrom

[14]CV – *Constant Voltage* (Konstantspannung): ein Zielkriterium, bei der mit variablen Belastungsströmen ein Spannungspegel auf einem konstanten Wert gehalten wird.

reduziert und die Entladung des Systems beginnt nach demselben Verfahren. Das Strom- und Temperaturprofil ist in Abb. 2.8 dargestellt.

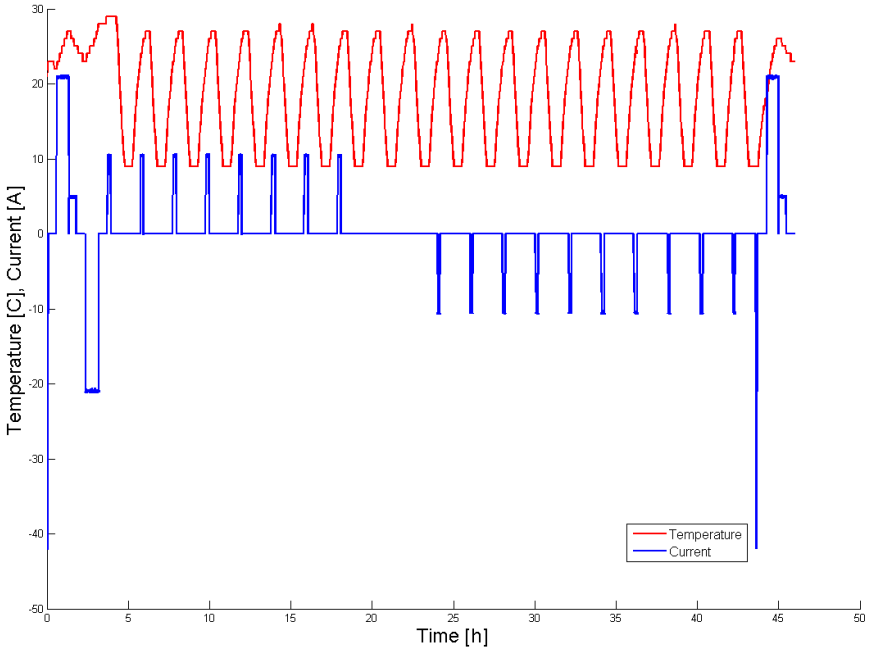

Abbildung 2.8: Das generierte Strom- und Temperaturprofil vom Test OCV-Steps

Verglichen wird das Testergebnis vom Prüfstand mit jenem in der Simulationsumgebung. Aus Zeitgründen wurde die Anzahl der Komplettsimulationsdurchläufe auf zwei begrenzt, da der Entwicklungsrechner die Simulationsdauer auf ca. 6000s begrenzte, aber ein kompletter Durchlauf 50h in Anspruch nahm. Ermöglicht wurde der Vorgang durch den Algorithmus aus `simulateExtensiveDataPackage` (siehe Abschnitt 2.2.6). Zusätzlich werden zur Demonstration der Parameterreduktion die Temperaturstützstellen über 20° C im originalen Parametersatz entfernt und der Fahrzyklus nochmal gestartet.

Abb. 2.9 zeigt die vollständige Simulation des Tests *OCV-Steps* am Beispiel der SOC-Berechnung (blaue Linie). Zum Vergleich wird der Prüfstands-Testverlauf (rote Linie) in der selben Abbildung gezeigt. Aufgrund der langen Testdauer wird die Übernahme der zuvor gespeicherten Zustände und der notwendigen NVM-Werte, einer jeden Teilsimulation in die darauf folgende, deutlich erkennbar. Für die Kapa-

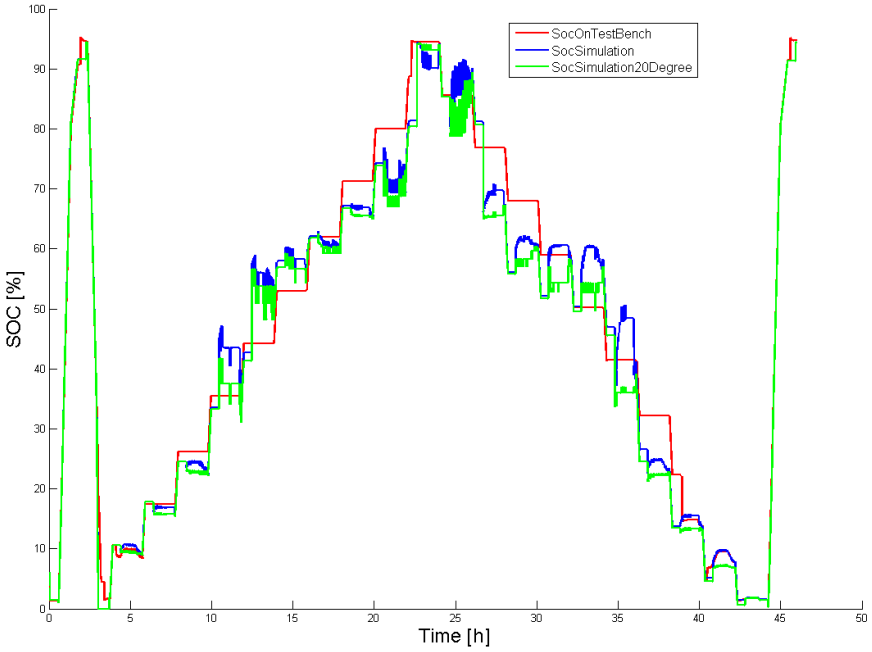

Abbildung 2.9: Vergleich von Simulation und Prüfstandsergebnis der Ladezustandsberechnung im Test *OCV-Steps*

zitätsbestimmung am Beginn und am Ende des Tests liefert die Simulation sehr gute Ergebnisse, verglichen mit den Prüfstandsergebnissen. Wie in Abschnitt 1.5.3.9 beschrieben, basiert dieses Verfahren auf Stromintegration, die in der Simulation und am Prüfstand auf die selbe Weise berechnet wird.
Die SOC-Berechnung in der Simulationsumgebung weicht aber deutlich sichtbar in den Ruhephasen vom Referenzsignal des Prüfstandes ab. Ein Grund für dieses Simulationsverhalten ist im Abschnitt 2.3.1.3 beschrieben, welches auf die Generierung des durchschnittlichen Zellspannungssignals verweist. Im realen System kann der tatsächliche Wert davon jedoch weit abweichen.
Einen weiteren Grund der Signalabweichungen in Abb. 2.9 liefert auch die höhere Anzahl der SOC-Korrekturen in der Simulation, dargestellt in Abb. 2.10. Die Möglichkeiten, den SOC zu korrigieren, werden laut Abschnitt 1.5.4.3 von den Fehlerwert des `SocVb` beeinflusst. Diese Berechnung zeigt aber ein sehr sensibles Verhalten, wenn die beeinflussenden Eingangssignale geringe Abweichungen von jenem am Prüfstand zeigen (Signalabweichung in Abb. 2.11 anhand eines `SocVb`-Fehlersignals dargestellt). Abweichungen können z.B. bei der Generierung eines fiktiven Stromsensor-Grundrauschens entstehen (siehe Abschnitt 2.3.1.2), oder liegen einer ungleichen

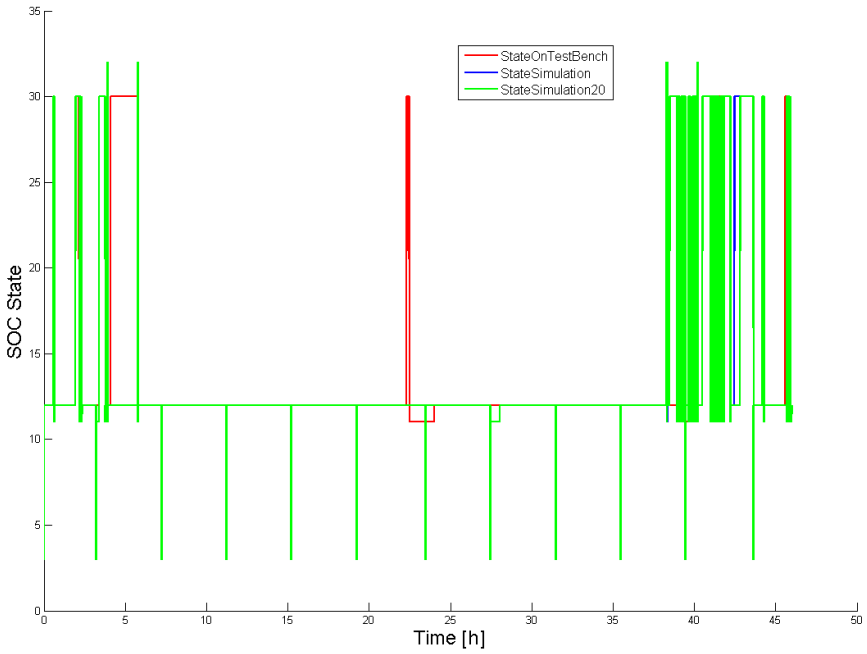

Abbildung 2.10: Der Korrekturstatus von *OCV-Steps*

Temperaturverteilung im Batteriesystem am Prüfstand zugrunde.
Die Simulation auf Basis eines reduzierten Parametersatzes war in gleicher Weise erfolgreich (grüne Linie in Abb. 2.9), wobei während dieser SOC-Berechnung eine weitere Zustandsabweichung erkennbar wird. Gut erkennbar sind SOC-Abweichungen im Vergleich zum simulierten SOC-Signal (blaue Linie), im mittleren SOC-Bereich zwischen $30 - 60\%$. Diese sind mit dem Eintreten in den nicht parametrierten SOC-Bereich bei Betriebstemperaturen über 13° C erklärbar, da die SOC/OCV-Kennlinien im mittleren SOC-Bereich weiter voneinander abweichen, als jene in den Randbereichen (Kennlinien über die Temperaturbereiche in Abb. 1.11). In diesem Fall werden, in Abhängigkeit des OCVs, nur SOCs bei der Temperatur bis 13° C für die Zustandsberechnung herangezogen. Auf das Korrekturverhalten hat diese Parameterreduktion kaum Auswirkungen (Vergleich in Abb. 2.10)

2.4.2.3 Test SOC-Variation

Der Test *SOC-Variation* fährt unter Raumtemperaturbedingungen den SOC von 20% an, um nach einer kurzen Pause diesen wieder auf 10% zu reduzieren. Ein Vorgang, um das weitere Spannungsverhalten auf der Entladekennlinie des OCVs

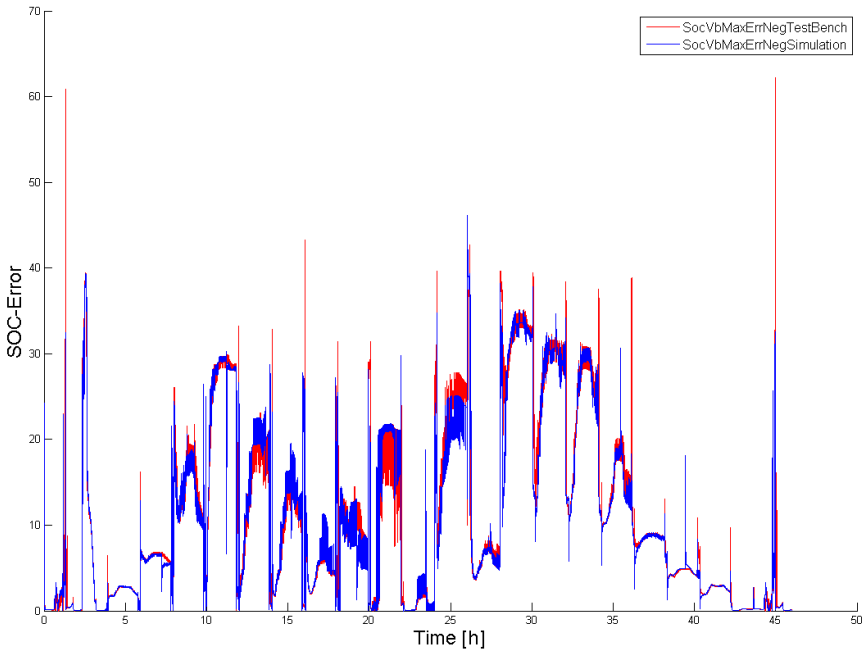

Abbildung 2.11: Das Fehlersignal des `SocVbMax` in negative Richtung

bei `Hyst`= 0 zu betrachten. Bei SOC 10% wird das System auf 0° C Umgebungstemperatur abgekühlt, um den Wechsel auf die OCV-Kennlinie dieses Temperaturbereichs zu erzwingen. Im Anschluss werden zwei Strompulse gefahren, zuerst in Entlade- dann in Laderichtung, um die Spannungsreserve bei dieser Temperatur zu zeigen. Wichtig dabei ist, dass die SOP-Grenzen der Stromamplitude nicht überschritten werden. Nach einer kurzen Pause soll das System komplett entladen werden, um denselben Test noch einmal zu fahren, wobei in diesem Fall auf die Abkühlung verzichtet wird. Dieser Test soll sich anschließend wiederholen, mit der Ausnahme, dass zuerst der SOC auf 45% geladen wird, um auf 35% entladen zu werden. Auch die Höhe der Strompulse zur Überprüfung der SOP- Limits sind dieselben (das generierte Stromprofil ist in Abb. 2.12 dargestellt).

Der Test *SOC-Variation* wurde mit Hilfe der generierten Prüfstandsdaten in der Simulationsumgebung in drei verschiedenen Varianten simuliert.

- Im ersten Fall wird der Testzyklus ohne Einschränkung simuliert.
- Im zweiten Fall werden die Temperaturstützstellen unter 10° C eliminiert.

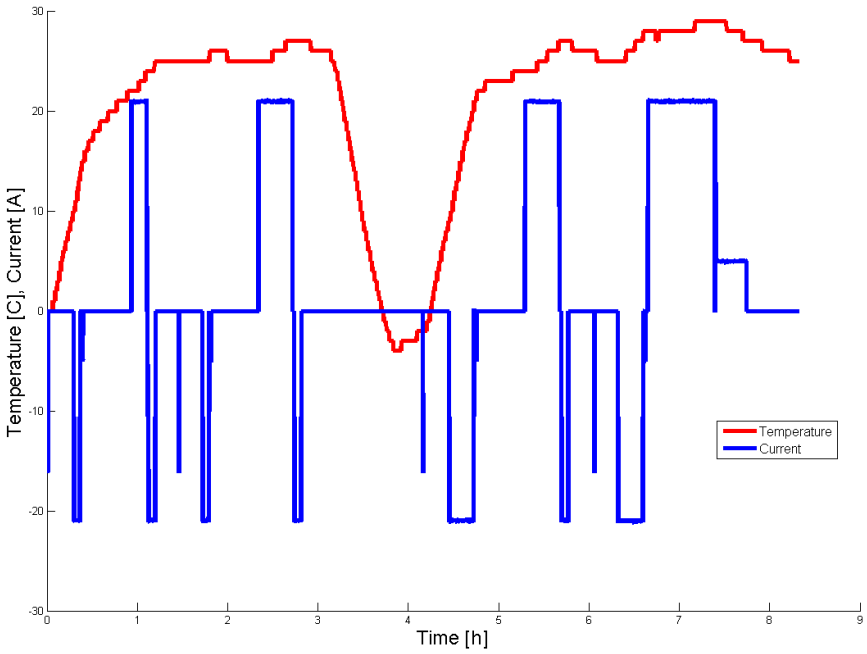

Abbildung 2.12: Das generierte Strom- und Temperaturprofil vom Test *SOC-Variation*

- Im dritten Fall wird der Umfang der SOC-Stützstellen reduziert, indem von SOC 4% an, nur jede dritte Stützstelle im Parametersatz belassen wird.

Das Ergebnis der Ladezustandsberechnung ist in Abb. 2.13 zu sehen. Ein unterschiedlicher Verlauf des SOCs am Prüfstand und in der Simulation ist erkennbar, bspw. nach ca. 1000s, wo der SOC in der Simulation auf über 20% springt. Das passiert zu jener Zeit, wo das System am Prüfstand auf unter 0° C gekühlt wird. Es ist anzunehmen, dass durch die ungleiche Temperaturverteilung am realen Batteriesystem der SOC stärker beeinflusst wird. Der SOC der Simulation wird auf Basis der gemessenen Systemtemperatur errechnet, die einen Durchschnittstemperaturwert des Systems beschreibt, dargestellt in Abb. 2.12. Der SOC-Sprung der drei Simulationen nach einer Simulationsdauer von ca. 3,3 Stunden kann nicht nachvollzogen werden.

Aufgrund des unplausiblen SOC-Sprungs in jeder der drei Simulationen kann kein genauer Vergleich zwischen den Prüfstands-Test und den generierten Simulationsergebnissen erfolgen. Im Ergebnis wird aber sichtbar, dass die zweite Simulation (grüne Linie) auf Basis eines reduzierten Parametersatzes eine geringe Abweichung

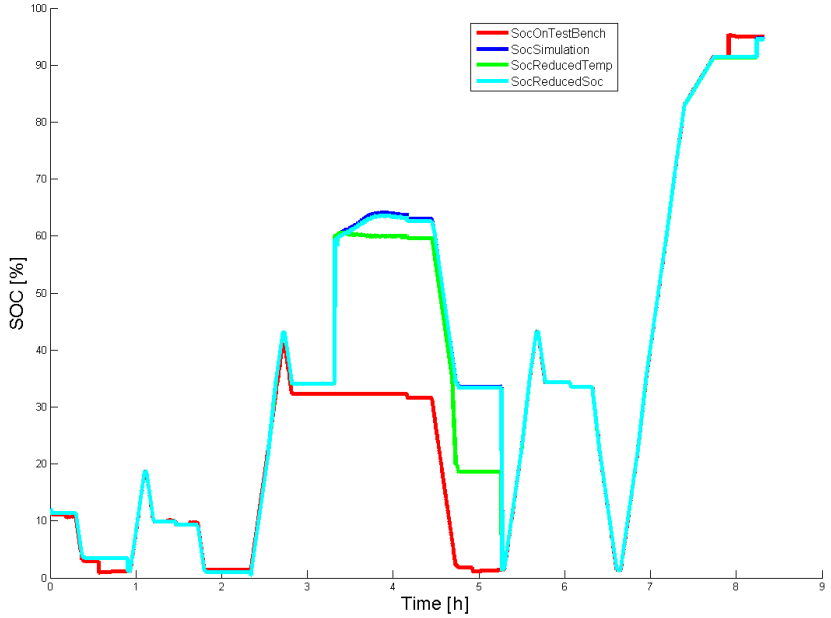

Abbildung 2.13: Vergleich von Simulation und Prüfstandsergebnis der Ladezustandsberechnung im Test *SOC-Variation*

des simulierten SOC-Verlaufs beschreibt. Der dritte Fall weist hingegen kaum zusätzliche Abweichungen auf (Linie in Cyan). Ein Grund dafür stellt die Reduktion im steilen Verlauf des SOCs dar, worin die fehlenden Werte linear interpoliert werden und somit den Verlauf der ursprünglichen SOC/OCV-Kennlinie in moderater Weise nachbilden. Das Korrekturverhalten unterscheidet sich wiederum vom Prüfstandsergebnis (siehe Abb. 2.14), kann aber in diesem Fall nicht als primärer Grund der Signalabweichung betrachtet werden.

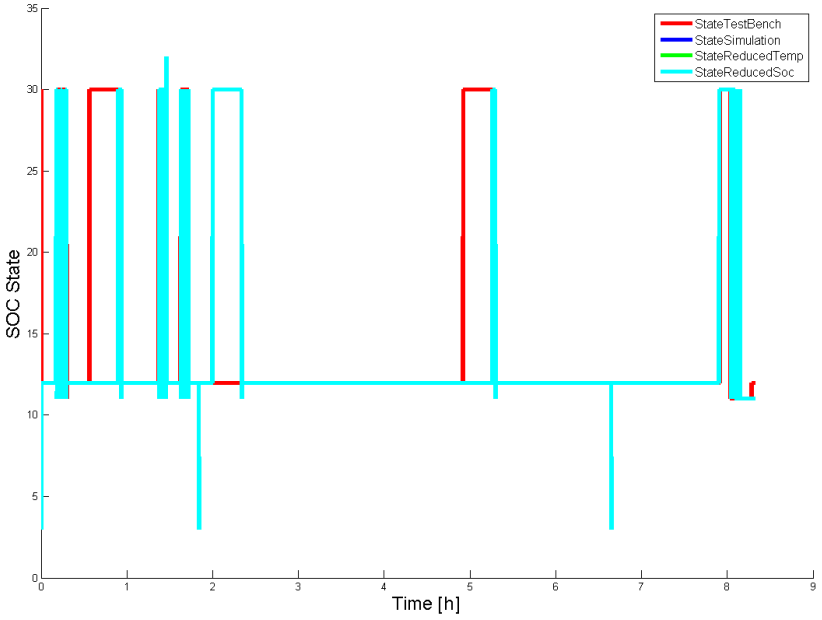

Abbildung 2.14: Der Korrekturstatus von *SOC-Variation*

3

Schlussbemerkung

3.1 Fazit

Unter der Voraussetzung, dass die Software des Batteriesystems am Prüfstand und das BMS in der Simulationsumgebung die selbe Versionsnummer besitzen, konnte beim Funktionsvergleich mit dem selben Ergebnis gerechnet werden. Dennoch blieb das Ergebnis der Funktionsanalyse hinter seinen Erwartungen zurück. Der Grund dafür liegt in der genaueren, realen Systembedatung am Prüfstand. Davon betroffen sind der Durchschnittswert der Zellspannungen, die Signale der Stromsignal-Abweichungen und die reale Temperaturschwankung der Zellen im Batteriesystem. Diese Signale wurden für die Simulation vereinfacht entworfen und im Modell implementiert, führen aber genau wegen ihrer Vereinfachung bspw. zu attraktiveren SOC-Korrekturmöglichkeiten, die in weiterer Folge die Systemzustände in unterschiedlicher Form verlaufen lassen.
Generell kann die in der Aufgabenstellung geforderte Verwendung der Simulationsumgebung am bereitgestellten Entwicklungsrechner nicht empfohlen werden. MATLAB/Simulink® sollte bei größer anfallenden Datenmengen im Bereich der Quell- und der Ergebnissignale mehr als 2 GB RAM Arbeitsspeicher zur Verfügung stehen. Somit ist die Simulation am Entwicklungsrechner zwar möglich, gilt aber als zeitaufwändig und nicht besonders komfortabel.

3.2 Zusammenfassung

In dieser Arbeit wurden die Simulation, daraus die Ergebnisaufbereitung und die Manipulation der Parameter einer modellbasierten PHEV-Serienbatteriesteuerungssoftware ermöglicht. Grundlage für die Realisierung der Arbeit bildete ein vorangegangenes Seminarprojekt, in dem ein Zellmodell in einer eigens dafür erstellten Entwicklungsumgebung die Eingangssignale für eine frühe Version einer PHEV-Steuerungssoftware in modellbasierter Form liefern soll. Die Zusammenschaltung sollte, in vereinfachter Form, einem HIL-Prüfstand nachempfunden sein.
Die Steuerungssoftware (BMS), sowie eine Nachbildung eines Verbandes von seriellen Lithium-Ionen Zellen, stehen als MATLAB/Simulink® Modell zur Verfügung. Die einzelnen Module der Steuerungssoftware unterscheiden sich zwischen den performanten S-Functions und den transparenten Simulink-Modulen. Das ebenso transparente Zellmodell liefert, alternativ zur direkten Bedatung des BMS mit Prüfstandsdaten, unterschiedliche Quellsignale zur Laufzeit. Somit wird es in Simulink® möglich, Betriebszustände des Zellmodells mit Hilfe des BMS zu errechnen, wie es am realen Batteriesystem der Fall ist.

Die Aufgabenstellung dieser Arbeit setzt das Ziel, die Ergebnisse der Simulation am Entwicklungsrechner, den Ergebnissen von Tests am Prüfstand, möglichst genau anzunähern. Signalpakete von verschiedenen Fahrzyklen, die am realen Batteriesystem vom Prüfstand aufgezeichnet wurden, liefern zum einen die Quellsignale für die Simulation, zum anderen beinhalten sie auch die Signalverläufe verschiedener Batterie-Zustände, die als Vergleichs-Referenz für die Simulation dienen.
Zusätzlich soll die Funktion geschaffen werden, den Parametersatz der Software an eingeschränkte Betriebsbedingungen anpassen zu können. Eine eingeschränkte Betriebsbedingung kommt im vorliegenden Fall zustande, wenn ein unvollständig charakterisierter Parametersatz einer Zelle zum Einsatz kommt, bspw. wenn man in verkürzter Zeit, durch eingeschränkte Zellcharakterisierungstests, einen Zellprototypen mit einer Software in einem Batteriesystem betreiben will.
Damit die Anforderungen der Aufgabenstellung erfüllt werden können, waren eine Reihe von Vorbereitungen, die Implementierung von mehreren Behelfen für die Initialisierung, die Ergebnisdarstellung und Maßnahmen zur Simulationsoptimierung notwendig.
Werden einige Abweichungen der Genauigkeit im Simulationsergebnis in Kauf genommen, kann anhand des Simulationsverhaltens durchaus auf das reale Batterieverhalten geschlossen werden. Gleiches gilt für die Simulation des BMS auf der Basis eines vorher reduzierten Parametersatzes. Die abweichende Systemfunktionalität kann vorausgesehen werden und bietet unter Umständen die Möglichkeit, als Vorlage für eine im Funktionsumfang reduzierte Steuerungssoftware von Zell-Prototypen zu dienen.

3.3 Ausblick

Für die zukünftige, weiterführende Verwendung des BMS in der Simulationsumgebung wird sich der Fokus auf die Erweiterung folgender Punkte konzentrieren:

- Der B-Muster-Stand des in dieser Arbeit verwendeten BMS wurde zum C- und D-Muster weiterentwickelt. Um die Simulink®-Modelle und S-Functions dieser Musterstände in der Simulationsumgebung zu betreiben, muss das Interface des BMS angepasst und ggf. weitere Eingangssignale aus dem Prüfstandsdatensatz bereitgestellt werden.
- Die fehlenden Systemfunktionalitäten (Kapazitätsbestimmung und Balancing) (siehe deren Modulbeschreibungen in Abschnitt 1.5.4.6 und 1.5.4.7), wurden im Rahmen der weiterentwickelten Musterstände hinzugefügt und müssen bei weiteren Funktionsanalysen in der Systemfunktion berücksichtigt werden.

- Ebenso ist im weiterentwickelten BMS die Funktion des SOH enthalten. Davon ist auch die Parametrierung des VCMs betroffen, wie bei der Beschreibung des VCMs in Abschnitt 1.5.4.2 angedeutet. Bei einer Parameterreduktion in dieser Entwicklungsstufe ist der Einfluss des SOH zu berücksichtigen.
- Das ursprünglich gesetzte Ziel dieser Arbeit, mit einem speziell reduzierten Parametersatz Software für den Betrieb von Prototypen-Zellen zu generieren, konnte aus Zeitgründen nicht erreicht werden. Dennoch wurde in dieser Arbeit die Funktionalität der Parameterreduzierung umgesetzt, aus deren Resultat in weiterer Folge eine Steuerungssoftware mit Hilfe von TargetLink® erstellt werden kann.

Literaturverzeichnis

[ABK08] Jonn Axsen, Andy Burke, and Kenneth S Kurani. Batteries for Plug-in Hybrid Electric Vehicles (PHEVs): Goals and the State of Technology circa 2008. Technical report, 2008. 1.3.2

[Bel12] Mary Bellis. http://inventors.about.com/od/estartinventions/a/History-Of-Electric-Vehicles.htm, 2012. 1.1

[Bot12] Dipl.-Chem. Patrick Bottke. *Primäre und wiederaufladbare Lithium-Batterien*. LU Technische Chemie, 2012. 1.4, 1.4, 1.4

[BS10] Ratnakumar V. Bugga and Marshall C. Smart. Lithium Plating Behavior in Lithium-Ion Cells. *ECS Transactions*, 25(36):241–252, 2010. 1

[CH08] M. Coleman and W.G. Hurley. An Improved Battery Characterization Method Using a Two-Pulse Load Test. *IEEE Transactions on Energy Conversion*, 23(2):708–713, June 2008. 1.6.3, 1.28, 1.6.3

[Cli09] Climatelab. http://climatelab.org/Plug-in_Hybrid_Electric_Vehicles, 2009. 1.2

[CLZH07] Martin Coleman, Chi Kwan Lee, Chunbo Zhu, and William Gerard Hurley. State-of-Charge Determination From EMF Voltage Estimation : Using Impedance , Terminal Voltage , and Current for Lead-Acid and Lithium-Ion Batteries. *IEEE Transactions on Industrial Electronics*, 54(5):2550–2557, 2007. 1.3.3, 1.6.4

[Doc10] Stefan Doczy. Entwicklung und Test von Batteriesystemen für Nutzfahrzeuge, 2010. 1.3.5

[ECKF11] Markus Einhorn, Valerio Conte, Christian Kral, and Jurgen Fleig. Comparison of electrical battery models using a numerically optimized parameterization method. *IEEE Vehicle Power and Propulsion Conference*, pages 1–7, September 2011. 1.5.4.2, 1.6.2, 1.6.4

[ECKM11] Markus Einhorn, Fiorentino Valerio Conte, Christian Kral, and Senior Member. Comparison , Selection and Parameterization of Electrical Battery Models for Automotive Applications. *IEEE Vehicle Power and Propulsion Conference*, (c):1–9, 2011. 1.6.2, 1.6.4

[Ele12] Electropeadia. http://www.mpoweruk.com/lithium_failures.htm, 2012. 1.4

[Emo11] EmobilityWeb. http://www.emobility-web.de/articles/markt-und-ueberblick/197/historie-das-erste-elektroauto-der-welt-faehrt-wieder, 2011. 1.1

[GBF10] Reza Ghorbani, Eric Bibeau, and Shaahin Filizadeh. On Conversion of Hybrid Electric Vehicles to Plug-In. *IEEE Transactions on Vehicular Technology*, 59(4):2016–2020, May 2010. 1.3.1

[Ha12] Hybrid-auto.info. http://www.hybrid-autos.info/Ueberblick-verschiedener-fahrzyklen.html, 2012. 1.6.2

[HLS11] Ari Hentunen, Teemu Lehmuspelto, and Jussi Suomela. Electrical battery model for dynamic simulations of hybrid electric vehicles. *IEEE Vehicle Power and Propulsion Conference*, pages 1–6, September 2011. 1.3.3, 1.5.3.8, 1.5.3.9, 1.6.1, 1.26, 1.6.1

[JH08] Marijn R Jongerden and Boudewijn R Haverkort. *Which battery model to use ?* PhD thesis, 2008. 1.3.3

[JW06] Andreas Jossen and Wolfgang Weydanz. *Moderne Akkumulatoren richtig einsetzen*. Ubooks Verlag, 2006. 1.1, 1.1, 1.3.3, 1.3.7, 1.4, 1.4, 1.4, 1.4, 1.4, 1.4

[Kle09] Ulrich Kletterer, Karl, Möst. Lithium-Ionen Batterien: Stand der Technik und Anwendungspotenzial in Hybrid-, Plug-In Hybrid- und Elektrofahrzeugen. *Wissenschaftliche Berichte FZKA 7503*, 2009. 1.1, 1.1, 1.3.1, 1.3.1, 1.4, 1.3.1, 1.5, 1.3.2, 1.7, 1.4, 1.5.2

[Mat12] Mathworks. http://www.mathworks.com/matlabcentral/newsreader/view_thread/52570, 2012. 2.2.6, 79

[MSB12] MSBS. Kundendokumentation der Li-Ion Antriebsbatterie der MAGNA Steyr Battery Systems, 2012. 1.3.6, 1.5.1, 1.5.3.2, 1.5.4.3, 2.1.4.5

[RSKN05] V. Rao, G. Singhal, a. Kumar, and N. Navet. Battery model for embedded systems. *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 105–110, 2005. 1.3.3, 1.6

[She63] C. M. Shepherd. Theoretical design of primary and secondary cells. *Electrochemistry Branch Chemistry Division*, 1963. 1.3.3, 1.6

[SMW11] Arash Shafiei, Ahmadreza Momeni, and Sheldon S Williamson. Battery Modeling Approaches and Management Techniques for Plug-in Hybrid Electric Vehicles. *IEEE Vehicle Power and Propulsion Conference*, 2011. 1.3.3, 1.3.3, 1.5.4.2

[TDD07] Olivier Tremblay, Louis-A. Dessaint, and Abdel-Illah Dekkiche. A Generic Battery Model for the Dynamic Simulation of Hybrid Electric Vehicles. *IEEE Vehicle Power and Propulsion Conference*, pages 284–289, September 2007. 1.3.3

[WBBR99] Henning Wallentowitz, Jan-Welm Biermann, Ralf Bady, and Christian Renner. Strukturvarianten von Hybridantrieben. *VDI-Tagung "Hybridantriebe", 25./26. 02.1999, München*, 1999. 1.3

[Wik12] Wikipedia. http://de.wikipedia.org/wiki/Lithium-Ionen-Akkumulator, 2012. 1.8

[Wik13] Wikipedia. http://de.wikipedia.org/wiki/NEFZ. 2013. 1.6.2, 2.4.2.1

[YGB08] Stephen Yurkovich, Yann Guezennec, and Raffaele Bornatico. Model-based calibration for battery characterization in HEV applications. *American Control Conference*, pages 318–325, June 2008. 1.6.3

[ZC10] Hanlei Zhang and Mo-yuen Chow. On-line PHEV Battery Hysteresis Effect Dynamics Modeling. *IECON - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 1844–1849, 2010. 1.5.3.7, 1.5.3.10, 1.16, 1.6.4, 1.29

[Zha10] Hanlei Zhang. Comprehensive Dynamic Battery Modeling for PHEV Applications. *IEEE Power and Energy Society General Meeting*, pages 1–6, 2010. 1.1, 1.5.3.7, 1.5.4.2, 1.6.1, 1.27, 1.6.2

A

Anhang

A.1 Listings

```
function cycleName = selectionDialog(path,topMsg)
    if exist(path)
        cycles = dir(path);
    else
        disp('Path does not exist!!!');
        return;
    end
    numCycles = size(cycles);

    if numCycles == 0
        disp('No drivng cycle found!!!');
        return;
    end
    cycleName = [];
    cycleNames = {};

    for cycleCount = 3:numCycles(1) %Ab dem 3. beginnen, um ./ und ../
        zu ueberspringen
        tempName = cycles(cycleCount).name;
        cycleNames{cycleCount-2} = cycles(cycleCount).name;  %
            cycleNames mit 1 zu nummerieren beginnen
    end
    [sel_idx, ok] = listdlg('Name','Select driving cycle ...',...
                            'SelectionMode','single',...
                            'ListString', cycleNames, ...
                            'ListSize', [600 500]); % Hoehe und Breite
                                des Auswahlfensters
    disp(topMsg);
    if ok,
        if strcmp(cycleNames{sel_idx}(end-3:end),'.mat')
            cycleName = strcat(cycleNames{sel_idx});
        else
            cycleName = strcat(cycleNames{sel_idx}, '.mat');
        end
      disp(['Select driving cycle   : ' cycleName]);
    else
      disp('Function aborted ...');
      return;
    end

end
```

Listing A.1: Der Quellcode aus `selectionDialog`

```
function [time  value] =  interpolate_input_20ms(input_time,
    input_value)

input_curr_time = input_time- input_time(1);    %Offset fuer Berechnung
     abziehen

input_curr_30 = input_value;

time_20 = 0:0.02:input_curr_time(end);

input_curr_20 = interp1(input_curr_time,input_curr_30,time_20,'linear')
    ;

time = (time_20+input_time(1))'; %Offset fuer Berechnung wieder
    hinzufuegen
value = input_curr_20';
```

Listing A.2: Der Quellcode aus `interpolate_input_20ms`

```
function output = replaceBlock(old, Mdl, new, Bib)
old = char(old);
Mdl = char(Mdl);
new = char(new);
Bib = char(Bib);
oldRef = find_system( Mdl, 'findall' , 'on' ,'LookUnderMasks','all','
    Name',old);
numberOfRepalceableBlocks = length(oldRef);
try
    for i=1:numberOfRepalceableBlocks
        newRef = find_system( Bib, 'findall' , 'on' ,'LookUnderMasks','
            all','Name',new);
        posOld = get_param(oldRef(i),'Position');
        pathOld = getfullname(oldRef(i));
        pathNew = getfullname(newRef);
        delete_block(oldRef(i));
        b1 = add_block(pathNew,pathOld);
        set_param(b1,'Position',posOld,'Name',new);
    end
catch
   disp('replaceBlock failed!!!');
end
output = numberOfRepalceableBlocks;
end
```

Listing A.3: Der Quellcode aus `replaceBlock`

```
function [startIndex, endIndex, emptyStruct, outStruct] =
    splitInputStruct(inStruct,splitFactor,packageNum)

    Field = fieldnames(inStruct);
    emptyStruct = false;  % Struct leer?
    for iField = 1:length(Field)
       Data = inStruct.(Field{iField});
       FieldName = Field{iField};
       if isa(Data, 'struct')  % Wenn Ast, sonst Blatt

            [startIndex, endIndex, emptyStruct outStruct.(FieldName)] =
                splitInputStruct(Data,splitFactor,packageNum); % Name
                des Asts uebernehmen

       else
            dataLength = length(Data);
            startIndex = ceil((dataLength/splitFactor)*(packageNum-1));
                %Start Index aufrunden - beim ersten File auf 1 setzen
            if startIndex < 1
                startIndex = 1;
            end
            endIndex = floor((dataLength/splitFactor)*(packageNum)); %
                End Index abrunden - am Ende des zu splittenden
                Datensatzes aufs Ende setzen
            if (dataLength-endIndex) < 5
                endIndex = dataLength;
            end
            outStruct.(FieldName) = Data(startIndex:endIndex); %
                begrenzte Daten in neue Struktur uebernehemn
            emptyStruct = true;

       end

    end
```

Listing A.4: Der Quellcode aus `splitInputStruct`

```
function vecApprox = functionSmoothing(data,windowSize)

% Start u. Endpunkt fuer Linearisierung an den Raendern (Initial
    Condition)
startPoint = mean(data(1:10));
endPoint = mean(data(end-10:end));

% Funktion Filtern und anschl. am Beginn linear zu Startwert fuehren
dataNew = filter(ones(1,windowSize)/windowSize,1,data);
```

```
dataNew(1:windowSize) = linspace(startPoint,dataNew(windowSize),
    windowSize);

% Datensatz Umkehren und filtern, dieses mal zum Endpunkt linearisieren
dataRev = data(end:-1:1);
dataRevNew = filter(ones(1,windowSize)/windowSize,1,dataRev);
dataRevNew(1:windowSize) = linspace(endPoint,dataRevNew(windowSize),
    windowSize);

% Umgekehrten Datensatz wieder richtig stellen
dataRevNew = dataRevNew(end:-1:1);

% Ergebnis aus Mittelwert zw. gefilterten und umgekehrt gefilterten
% Datesatz bilden
vecApprox = (dataNew + dataRevNew)/2;

end
```

Listing A.5: Der Quellcode aus `functionSmoothing`

```
------------------------------ demogui3.m ------------------------------
function x = demogui3(selector)
%DEMOGUI3 Demonstration GUI application which returns an output.

% written by Douglas M. Schwarz
% Eastman Kodak Company
% douglas.schwarz@kodak.com
% 16 May 1997

if nargin == 0
    selector = 'init';
end

switch selector
case 'init' % Initialize GUI application
    myname = mfilename;
    fig = figure('Position',[245 380 150 110],...
            'NumberTitle','off',...
            'Name',myname,...
            'CloseRequestFcn',[myname,' close'],...
            'Resize','off',...
            'WindowStyle','modal');
    info.count = 1;
    uicontrol('Style','pushbutton',...
            'Position',[20 40 50 20],...
            'String','Down',...
            'CallBack',[myname,' dec'])
    uicontrol('Style','pushbutton',...
```

```
            'Position',[80 40 50 20],...
            'String','Up',...
            'CallBack',[myname,' inc'])
    info.editbox = uicontrol('Style','edit',...
            'Position',[45 70 60 20],...
            'String',num2str(info.count),...
            'CallBack',[myname,' edit']);
    uicontrol('Style','pushbutton',...
            'Position',[50 10 50 20],...
            'String','OK',...
            'CallBack',[myname,' close'])
    set(fig,'UserData',info,...
            'HandleVisibility','callback')

    uiwait(fig)

    info = get(fig,'UserData');
    x = info.count;
    delete(fig)

case 'dec' % Decrement counter
    fig = gcbf;
    info = get(fig,'UserData');
    info.count = info.count - 1;
    set(info.editbox,'String',num2str(info.count))
    set(fig,'UserData',info)

case 'inc' % Increment counter
    fig = gcbf;
    info = get(fig,'UserData');
    info.count = info.count + 1;
    set(info.editbox,'String',num2str(info.count))
    set(fig,'UserData',info)

case 'edit' % Enter value in edit box
    fig = gcbf;
    info = get(fig,'UserData');
    newcount = sscanf(get(info.editbox,'String'),'%d');
    if ~isempty(newcount)
        info.count = newcount;
    end
    set(info.editbox,'String',num2str(info.count))
    set(fig,'UserData',info)

case 'close' % Close requested
    uiresume(gcbf)

```

```
end
```

Listing A.6: Das Programmierbeispiel nach dem Beitrag aus Quelle [Mat12], auf dessen Basis die GUIs dieser Arbeit entstanden sind

A.2 Lut-Bib

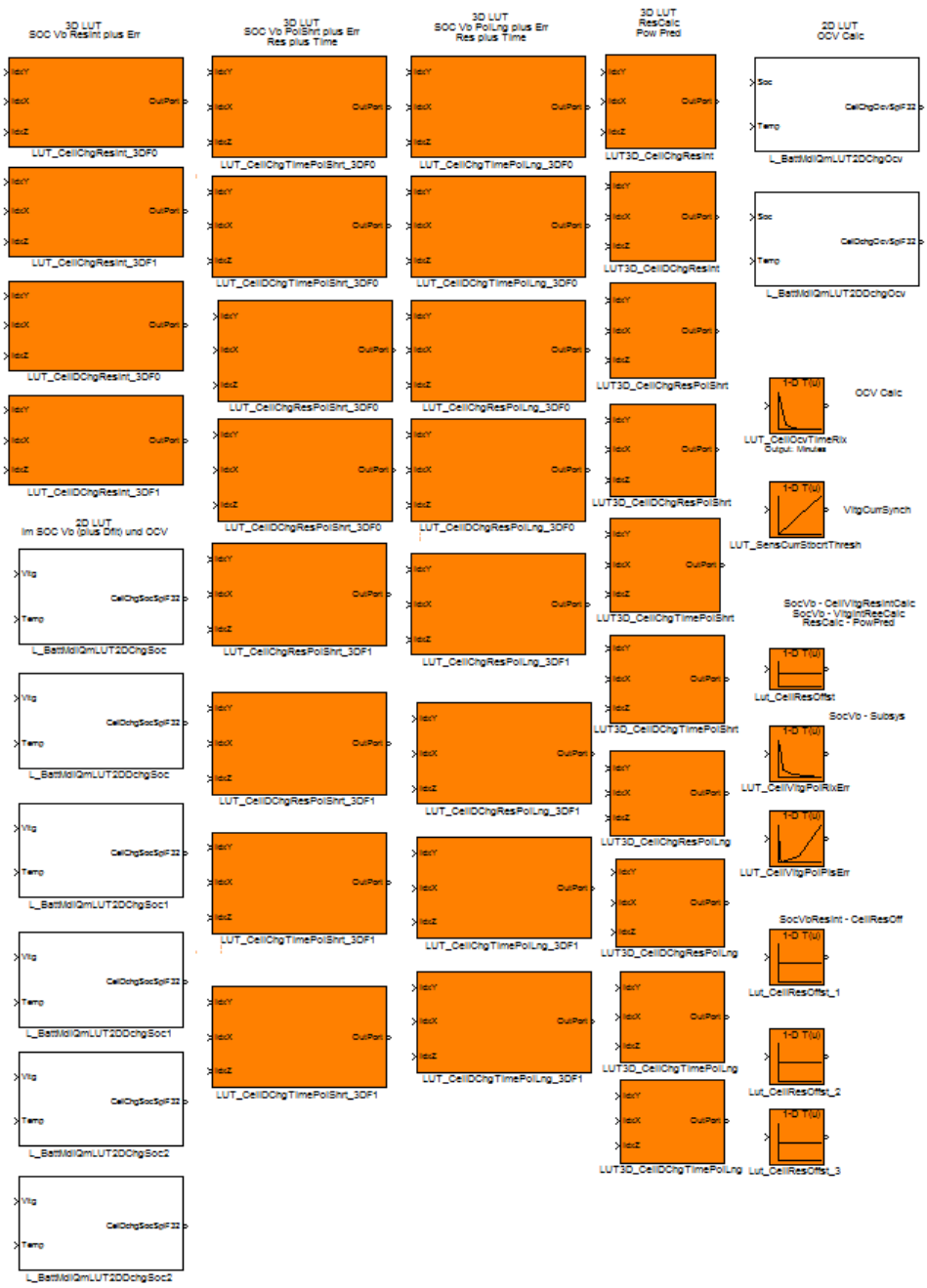

Abbildung A.1: Diese LUTs werden in vereinfachter Form für die Simulink®-Module zur Verfügung gestellt

A.3 Testskript

A.3.1 OCV-Steps

CI ... Coolant inlet
CV ... Constant voltage
CC ... Constant current
CP ... Constant power
C ... 21A

Test set points	
Temperature	
T1	23 °C
T2	10 °C
T3	30 °C
SOCx	
SOC1	10%
SOC2	20%
SOC3	30%
SOC4	40%
SOC5	50%
SOC6	60%
SOC7	70%
SOC8	80%
SOC9	90%
SOC10	100%

Capa Check

State	action	time	target		Break condition	StepCounter/ Loop counter
			Coolant inlet	Current / Voltage		
	Open new file for saving data					
1	**Discharge - Reset AhTB**		CI = T1	CC = 1C	$U_{cellMin}$< 2 V	
2	**Pause**	1900 sec	CI = T1			
3	**Charge**		CI = T1	CC = 1C	$U_{cellMax}$ > 3,5 V OR U_{BATT} > 420 V	
4	**Pause**	1900 sec	CI = T1			
	AhStep4=AhTB					
5	**Discharge - Reset AhTB**		CI = T1	CC = 1C	$U_{cellMin}$< 2 V	
	AhStep5=AhTB			ActCapa = abs(AhStep4 -AhStep5)		

OCV Steps

State	action	time	target		Break condition	StepCounter/ Loop counter
			Coolant inlet	Current / Voltage		
OCV Steps ***Loops: 10***	*Open new file for saving data*					
1.1	**Pause**	1800 sec	CI = T3			
1.2	**Charge**		CI = T3	CC = C/2	$U_{cellMax}$ > 3,6 V OR (AhTB/ActCapaAh)*100 == SOCx	
1.3	**Pause**	900 sec	CI =T3			
1.4	**Pause**	max. 3600 sec	CI =T2		TcellAvg = 10 °C, deltaTcell <3 °C	
OCV Steps ***Loops: 10***						
2.1	**Pause**	1800 sec	CI = T3			
2.2	**Discharge**		CI =T3	CC = C/2	$U_{cellMin}$< 2 V OR (AhTB/ActCapaAh)*100 == SOCx	
2.3	**Pause**	900 sec	CI =T3			
2.4	**Pause**	max. 3600 sec	CI =T2		TcellAvg = 10 °C, deltaTcell <3 °C	

Capa Check

State	action	time	target		Break condition	StepCounter/ Loop counter
			Coolant inlet	Current / Voltage		
	Open new file for saving data					
1	**Discharge - Reset AhTB**		CI = T1	CC = 1C	$U_{cellMin}$< 2 V	
2	**Pause**	1900 sec	CI = T1			
3	**Charge**		CI = T1	CC = 1C	$U_{cellMax}$ > 3,5 V OR U_{BATT} > 420 V	
4	**Pause**	1900 sec	CI = T1			