TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Realzeit-Computersysteme

Models and Interfaces for Distributed Control Systems

Matthias Kauer

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Gerhard Rigoll
Prüfer der Dissertation:	1. Prof. Dr. Samarjit Chakraborty
	2. Prof. Anuradha M. Annaswamy, Ph.D.

Die Dissertation wurde am 24.03.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 17.08.2017 angenommen.

Abstract

The goal of this thesis is to bridge the gap in digital control systems between high-level domain models for physical dynamics and implementation platforms. To that end, suitable interfaces linking physical and platform behavior are proposed for two setups. In the first, these interfaces improve guaranteed control performance under constrained communication. In the second, fast simulation and optimization are achieved for a class of switching-actuated systems.

Automatic control, the context of this work, is concerned with the regulation of dynamical systems. A control algorithm periodically adjusts a system's input signal to ensure a certain output behavior. These algorithms are typically designed in isolation, with a number of idealistic assumptions about their environment. Such assumptions include infinite numerical precision and zero delays on the hardware platform, as well as mathematically convenient input signals. Although approximately satisfied in many cases, they are not sufficiently analyzed in others.

Combining the design of control algorithms and their implementation platforms is challenging because their respective models deal with very different phenomenons. System descriptions and control algorithms typically express dynamics in continuous time, using differential equations. Digital platforms have many discrete aspects and comprise processor architectures, scheduling strategies and communication protocols, among other things. Interfaces are hence required that describe or abstract the behavior of the physical side in a form that can be taken into account during the design process of the digital side and vice versa.

Developing suitable and expressive interfaces that enable a co-design of control algorithm and digital platform is the focus of this thesis. To that end, we study two different setups. The first deals with the effects of communication delay on the control performance of quickly evolving systems. There, the requirements of each control algorithm are typically expressed as a combination of period and deadline. This is rather rigid and the more flexible interface introduced here leads to designs with higher guaranteed performance on identical hardware. The second setup considers charge transfer within battery packs. Each of the many individual cells in such a pack behaves somewhat differently, yet the aggregate charge is determined by the minimum charge among cells in the pack. For this reason, transferring charge and thus balancing cells improves efficiency. Because these transfers are actuated by frequent discrete switching, abstracting interfaces are crucial for efficient quantitative models. Both models and interfaces are proposed in this dissertation.

Communication and computation delay, as studied in the first setup, is the main link between rapidly changing dynamical systems and their implementation platform. This delay becomes non-negligible in complex distributed platforms that have been introduced to cope with the growing number of control applications. The systematic design processes, which have been and are being developed for larger systems, do still not include delay interaction in adequate detail,

however. As delay affects control performance while timing requirements also affect platform design in return, a co-design of control algorithm and implementation platform can improve the overall process quality and dramatically reduce testing efforts.

This is the promise of the *Cyber-Physical System* (CPS) paradigm. It is different from previous approaches like networked control systems where the implementation platform (or the network) is assumed to be fixed. By default, timing requirements are specified in terms of strict deadlines. These are formed with respect to the worst case which only rarely occurs. Designing towards that interface hence leads to guaranteed, but not necessarily to sufficient performance. The design framework presented here allows and quantifies occasional deadline misses with their effects on the physical system using formal verification. It can thus use shorter deadlines and ultimately achieves better performance guarantees. To make the approach comprehensive, automata-theoretic models for fixed priority scheduling are advanced to yield conservative results even with non-preemptive behavior. A delay-aware fault-tolerant control strategy further improves guaranteed performance.

In the second setup, internal charge transfer increases battery pack efficiency because usable charge is maximized when the charge of all cells is in balance. In this process, also referred to as active cell balancing, network timing is less critical because it is slow compared to the network and other control applications like anti-lock braking. Its actuation, however, intersects digital and analog behavior in a challenging way. Charge is transferred most efficiently by rapidly switching transistors that connect a temporary energy storage element, like an inductor, to the sender and receiver cell in alteration. This switching leads to non-differentiable transitions between the sender and receiver phase, interfering with common simulation techniques and strategy design. Often, it is hence entirely abstracted away in a lossy way. By contrast, the thesis at hand preserves detail and improves speed by deriving closed-form expressions for the individual transfer phases of a transmitting cell pair. This proposed reformulation leads to 1000 times faster simulation and lossless interfaces that specify long-term actuation. To evaluate transfers beyond minutes, this work consequently develops formulations that enable optimization techniques and speed up the aforementioned phase-based simulation by additional orders of magnitude.

The discussed setups share a number of characteristics. In particular, they treat digital control systems with a semantic gap between models for the physical dynamics and the implementation platform. The solutions we propose towards modeling, controlling, and optimizing them rely on the developed interfaces that bridge this gap. It is worth mentioning how these approaches relate to contributions in the domain of hybrid systems. These include both continuous and digital behavior in one monolithic model, limiting the system size. In timing analysis, the number of discrete states that model network behavior is prohibitively large for hybrid model checking tools. In case of active cell balancing, a hybrid model corresponds to the initial slow simulation we start out with. The techniques developed in this thesis integrate communication and actuation aspects tightly, improving accuracy and hence performance compared to state-of-the-art methods, while ensuring computational efficiency.

Kurzfassung

Das Ziel dieser Arbeit ist die Lücke zu schließen zwischen abstrakten, domänenspezifischen Modellen für physikalische dynamische Systeme und für Implementierungsplattformen. Zu diesem Zweck schlägt diese Arbeit geeignete Schnittstellen vor, die physikalisches und digitales Verhalten in zwei Aufbauten verbinden. Im ersten Aufbau führen diese Schnittstellen zu verbesserter garantierter Regelungsleistung unter eingeschränkter Kommunikation. Im Zweiten werden schnelle Simulation und Optimierung erreicht für eine Klasse von Systemen, die durch digitales Schalten gesteuert werden.

Die Regelungstechnik, die den Kontext für diese Arbeit bildet, beschäftigt sich mit der Regulierung von dynamischen Systemen. Ein Regelungsalgorithmus passt die Eingangssignale eines Systems periodisch an, um ein bestimmtes Ausgangsverhalten zu gewährleisten. Diese Algorithmen werden üblicherweise isoliert entworfen, mit verschiedenen idealisierenden Annahmen über ihre Umgebung. Zu diesen Annahmen gehören unendliche numerische Präzision und null Verzögerung auf der Hardwareplattform, aber auch mathematisch vorteilhaft gewählte Eingangssignale. Wenngleich solche Annahmen in vielen Fällen annähernd erfüllt sind, werden sie in anderen Fällen nicht ausreichend untersucht.

Das Design von Regelungsalgorithmen und ihren Implementierungsplattformen zu kombinieren ist schwierig, weil ihre jeweiligen Modelle mit sehr unterschiedlichen Phänomenen befasst sind. Systembeschreibungen und Regelungsalgorithmen drücken typischerweise die Dynamik in kontinuierlicher Zeit aus unter Verwendung von Differenzialgleichungen. Digitale Plattformen haben viele diskrete Aspekte und beinhalten unter anderem Prozessorarchitekturen, Schedulingstrategien und Kommunikationsprotokolle. Es werden daher Schnittstellen benötigt, die das Verhalten der physikalischen Seite in einer Form beschreiben oder abstrahieren, die während des Designprozesses für die digitale Seite berücksichtigt werden kann und umgekehrt.

Die Entwicklung geeigneter und ausdrucksstarker Schnittstellen, die ein Co-Design von Regelungsalgorithmus und digitaler Plattform ermöglichen, steht im Fokus dieser Dissertation. Zu diesem Zweck werden zwei verschiedene Aufbauten untersucht. Der erste beschäftigt sich mit den Effekten von Verzögerung bei der Kommunikation auf die Regelungsleistung von Systemen mit schnellem Eigenverhalten. Dort werden die Anforderungen jedes Regelungsalgorithmus typischerweise als Kombination von Periode und Deadline ausgedrückt. Das ist eher starr und die flexiblere Schnittstelle, die hier eingeführt wird, ermöglicht Designs mit höherer garantierter Leistung auf identischer Hardware. Der zweite Aufbau betrachtet Ladungstransfers innerhalb von Batteriepacks. Jede der vielen individuellen Zellen in einem solchen Pack verhält sich leicht anders, während die Gesamtladung bestimmt wird durch die Minimalladung unter den Zellen des Packs. Aus diesem Grund erhöhen Transfers und der so erzeugte Ladungsausgleich die Effizienz. Weil diese Transfers durch rapides, diskretes Umschalten getrieben werden, sind abstrahierende Schnittstellen entscheidend für die Erstellung von effizienten, quantitativen Modellen. Sowohl Modelle als auch Schnittstellen werden im Rahmen dieser Arbeit vorgeschlagen.

Verzögerungen bei der Übertragung oder bei der Berechnung, wie sie im ersten Aufbau untersucht werden, sind die Hauptverbindung zwischen sich schnell wandelnden dynamischen Systemen und ihrer Implementierungsplattform. Diese Verzögerung kann nicht mehr vernachlässigt werden bei komplexen,

verteilten Plattformen, welche eingeführt wurden, um mit der wachsenden Anzahl von Regelungsanwendungen umzugehen. Die systematischen Designprozesse, welche für große Systeme entwickelt wurden und werden, behandeln allerdings weiterhin nicht den Einfluss von Verzögerung in angemessenem Detail. Da Verzögerung die Regelungsleistung beeinflusst während Timinganforderungen umgekehrt auch das Plattformdesign beeinflussen, kann ein Co-Design von Regelungsalgorithmus und Implementierungsplattform die Prozessqualität insgesamt verbessern und den Testaufwand stark reduzieren.

Das ist das Versprechen des *Cyber-Physical System* (CPS) Paradigma. Es unterscheidet sich von früheren Ansätzen wie vernetzten Regelungssystemen, wo die Implementierungsplattform (oder das Netzwerk) als fest angenommen werden. Standardmäßig werden Timinganforderungen als strikte Deadlines angegeben. Diese werden gebildet mit Bezug auf den ungünstigsten Fall, welcher nur äußerst selten auftritt. Auf diese Schnittstelle zu designen führt daher zu garantierter, aber nicht notwendigerweise zu ausreichender Performance. Das Designframework, das hier vorgestellt wird, lässt gelegentliche Zeitüberschreitungen zu und quantifiziert sie mitsamt ihren Auswirkungen auf das physikalische System mit formaler Verifikation. Es kann daher kürzere Fristen verwenden und schlussendlich bessere Leistungsgarantien erreichen. Um den Anwendungsbereich für diesen Ansatz zu erweitern, werden automatentheoretische Modelle für Scheduling mit festen Prioritäten vorangebracht, so dass sie konservative Ergebnisse liefern auch wenn Nachrichten nicht unterbrochen werden können. Eine fehlertolerante Regelungsstrategie, die sich Verzögerungen bewusst ist, erhöht die garantiere Leistung weiter.

Im zweiten Aufbau erhöhen interne Ladungstransfers die Effizienz eines Batteriepacks, weil die nutzbare Ladung maximal wird, wenn alle Zellladungen ausgeglichen sind. In diesem Prozess, der auch als Active Cell Balancing bezeichnet wird, ist das Timing des Netzwerks weniger entscheidend, da er langsam ist verglichen mit dem Netzwerk und anderen Regelungsanwendungen, wie einem Antiblockiersystem. Seine Betätigung kreuzt allerdings digitales und analoges Verhalten auf herausfordernde Weise. Ladung wird am effizientesten übertragen durch schnell umschaltende Transistoren, die einen temporären Energiespeicher, wie eine Spule, abwechselnd mit Sender- und Empfängerzelle verbinden. Dieses Umschalten führt zu nicht-differenzierbaren Übergangen zwischen der Sender- und der Empfängerphase und behindert gängige Simulationsmethoden wie auch den Strategieentwurf. Es wird daher oft einfach wegabstrahiert auf verlustbehaftete Weise. Im Gegensatz dazu erhält die vorliegende Dissertation die Details und verbessert die Simulationsgeschwindigkeit, indem sie geschlossene Ausdrücke für die einzelnen Transferphasen herleitet. Diese vorgeschlagene Umformulierung ermöglicht 1000 mal schnellere Simulation und verlustfreie Schnittstellen, welche die Langzeitoperation spezifizieren. Um Transfers zu evaluieren, die über Minuten hinaus gehen, entwickelt diese Arbeit anschließend Formulierungen, die Optimierung ermöglichen und den erwähnten phasenbasierten Simulationsansatz um weitere Größenordnungen beschleunigen.

Die diskutierten Aufbauten teilen mehrere Eigenschaften. Insbesondere behandeln sie beide digitale Regelungssysteme mit einer semantischen Lücke zwischen den Modellen für die physikalische Dynamik und denen für die Implementierungsplattform. Die Lösungen, die zur Modellierung, Regelung und Optimierung vorgeschlagen werden, beruhen auf Schnittstellen, die diese Lücke überbrücken. Es soll auch erwähnt werden, in welchem Verhältnis diese Ansätze zu Beiträgen aus dem Bereich der hybriden Systeme stehen. Diese beinhalten sowohl kontinuierliches als auch digitales Verhalten in einem monolithischen Modell, was die Systemgröße beschränkt. In der Timinganalyse ist die Anzahl an diskreten Zuständen, die das Netzwerkverhalten modellieren, zu groß für hybride Model Checking Tools. Im Fall von Active Cell Balancing entsprechen hybride Modelle den initialen, langsamen Modellen, die hier verbessert werden. Die Techniken, die in dieser Dissertation entwickelt werden, integrieren Kommunikations- und Antriebsaspekte auf enge Weise und verbessern dadurch die Genauigkeit und somit die Performance im Vergleich zum Stand der Technik, während effiziente Berechnung gewährleistet bleibt.

Acknowledgements

This thesis is the result of my work in the embedded systems group (RP3) of TUM CREATE, a research collaboration of Technische Universität München (TUM) and Nanyang Technological University funded by Singapore's National Research Foundation under the CREATE program.

Many people deserve my gratitude for their support. First, I would like to thank my advisor, Samarjit Chakraborty from the Institute for Real-Time Computer Systems (RCS) at TUM, for his advice during the past four years. I consider myself very fortunate to have received your insights and encouragement during our meetings. Furthermore, the collaborations and the research stay you initiated for me are invaluable. At the same time, I also highly appreciate the guidance I have received from Martin Lukasiewycz and Sebastian Steinhorst, the postdocs and team leaders in RP3. We collaborated on virtually all my undertakings and I learned a great deal from you about verification, optimization, and software engineering.

This work contains two major parts: distributed control systems and active cell balancing. With regard to the first part, I would like to thank Reinhard Schneider and Dip Goswami from RCS for answering my incessant questions about timing analysis, real-time calculus, and control systems. With regard to the second part, my gratitude goes to Swaminathan Narayanaswamy and Arne Meeuw from RP3. You have significantly enhanced my understanding of electrical circuits and embedded programming. Many of the experiments from this thesis would also not have been possible without the hardware you built.

My gratitude also goes to Damoon Soudbakhsh and Anuradha Annaswamy from the Active-Adaptive Control Laboratory at the Massachusetts Institute of Technology. During our collaboration which related to both parts, you have taught me a lot about control systems, improving my intuition as well as my understanding of techniques like linear matrix inequality design, adaptive control, or model-predictive control. Thank you also, Dr. Annaswamy, for hosting me in your group in fall 2014 and for being my second reviewer.

I am also fortunate for the colleagues I have had during the last years. Our laughs, technology discussions, and political rants have made the time more than enjoyable. A special note goes to Philipp Mundhenk, Florian Sagstetter, and Peter Waszecki. I hope we can stay in touch and meet for gaming, BBQ, or just a beer. A special thank you also goes to Martin Geier, for help with IT and enlightening Linux discussions, Martin Schäfer, who introduced me to RCS, and Benedikt Dietrich, for help with projects I supervised. Another big thank you is in order for the administrative staff at MIT, TUM CREATE, and at RCS. You have been very helpful and supportive.

Finally, I particularly thank my friends, my parents, Waltraud and Herbert, and my sister Judith for their love, support, and continuous encouragement.

Contents

1 Introduction

The regulation of dynamical systems using feedback mechanisms has a long history. The first application arguably dates back to ancient Greece where the flow in water clocks was regulated for improved accuracy. Starting with the industrial revolution in the 19th century, centrifugal governors were widely used to control the inflow of steam engines. Alongside further similar contributions, the mathematical formalization of the field also began at that time.

Even though methods for system analysis in time domain like the solution of Ordinary Differential Equations (ODEs) and stability concepts were known, the lack of computation power made them infeasible in practice. Frequency domain analysis, by contrast, makes it easier to characterize a system from external measurements and the arithmetics it involves remain tractable for computation by hand. This made it the method of choice at that time. Only the arrival and subsequent massive growth of computing devices has increased the focus on time-based methods over recent decades. Numerically efficient and stable techniques were devised to test system stability, design optimal input signals, and reconstruct unobservable system states, among other applications.

In addition to a dramatic expansion of system analysis techniques, digital devices have also led to new actuation and controller implementation methods. By introducing computation power into the system itself, the class of digital control systems provides a lot of new options to the strategy designer. The main advantage is that calculation steps can be freely chosen and no longer require individual physical components that represent them. Digital actuation, on the other hand, significantly increases the efficiency of, e.g., electric motors or power supplies. While the addition of digital devices creates size, efficiency, and flexibility benefits over previous analog or mechanical implementations, the so-created interaction also changes the system behavior and may require adjustments in the design techniques.

Combining the behavior of digital implementation platform and physical control process into one holistic model is difficult because their respective models deal with very different phenomenons. The dynamics of the control process are typically expressed in continuous time,

using differential equations. In the digital platform, on the other hand, topics like processor architecture, scheduling strategy and communication protocol contain many discrete aspects. During the design process of the physical side, the behavior of the digital side must hence be abstracted according to a suitable interface and vice versa.

The selection of interfaces for digital control design involves a trade-off between modeling accuracy, validity area, and speed. In this context, the thesis at hand develops interfaces that are more expressive than the state of the art and hence lead to better designs but also remain computationally efficient to enable rapid iteration and optimization during the design phase. With this goal, we investigate two different setups. The first (Section 1.1) deals with the effects of communication delay on the control performance of quickly evolving systems. As the number of control applications keeps growing, the digital implementation platforms become increasingly distributed, creating contention on shared buses and processors. This calls into question several standard assumptions about the computation behavior and complicates the design task for the implementation platform. The second setup (Section 1.2) considers switching actuation and analyzes a family of circuits that transfer charge within battery packs for improved performance. In contrast to paradigms where switching is undesired, switching-actuated systems are controlled by adjusting the, typically brief, time periods for which they maintain certain discrete configurations. Charge transfer is particularly challenging since the averaging approach, which is typically employed to circumvent non-differentiable transitions and the associated modeling difficulties, becomes inaccurate as internal states evolve. These challenges are summarized in Table 1.1.

Table 1.1: *Overview of challenges*

	CPS co-design (Section 1.1)	Charge transfer (Section 1.2)
Physical	Fast control process	Cells with transfer circuitry
Digital	Controller-implementing hardware	Actuation by transistor switching
Challenge	Delay not negligible and varying	Input via timing variation
Interface	Timing specification	Conversion: current $\leftrightarrow$ timing
Gap	Cannot model platform characteristics, i.e., causes for delay	Conversion is voltage- and thus state-dependent

Both setups deal with a semantic gap between models for the physical dynamics and the behavior of a digital device. This gap interferes with accurate analysis of the system as a whole. Several approaches that lead to tighter integration for better performance, faster simulation at virtually identical accuracy, and efficient optimization are developed in the context of this thesis. Section 1.3 details how these contributions and the publications they have led to form the remainder of the work at hand.

1.1 Cyber-physical System (CPS) co-design

In digital control, a computing device is used as system controller to calculate input signals at runtime. Inexpensive microcontrollers are typical for this role, but Application-Specific In-

tegrated Circuits (ASICs) or desktop computers are also reasonable depending on the performance requirements. The controller in such a setup periodically evaluates the sensor values describing the system state and subsequently calculates the input signals for actuation at runtime. Forgoing analog components in that way provides many benefits, but it may also require certain adjustments to the analysis methods.

The controller cannot compute input signals continuously as that would correspond to infinite frequency. For this reason, input signals in digital control typically remain constant for a certain period; a case of digital control with a different actuation abstraction is discussed in Section 1.2. The standard periodic behavior of the computation suggests a transformation of the mathematical models that describe the physical world for easier analysis. Instead of calculating the continuous-time dynamics of such a digital control system directly, it is common to consider them only with respect to the discrete time steps that are implicitly defined by the sampling period of the controller. Many previous results for system analysis and design have been adjusted to systems described in this way, so-called discrete-time systems.

When the digital control framework was formed, ideal circumstances in military applications, space missions, and avionics have justified a number of idealistic assumptions about the underlying computation platform. The most common of these assumptions are negligible computation times, infinite numerical precision as well as zero communication delays between sensor, controller, and actuator. With a dedicated controller that has direct access to sensors and actuators, such assumptions are approximately satisfied and small deviations are unlikely to produce large errors. Such setups are expensive and increasingly rare, however, as digital control systems have become ubiquitous, even in cost-sensitive industries like automotive, home automation, and energy storage. In these industries, computation power is more limited and control applications are implemented in a distributed fashion. Each Electronic Control Unit (ECU) typically handles many control tasks and no longer has direct access to sensors and actuators. Instead, sensor readings are sent over a network, processed by one or more ECUs along the way, before finally arriving at the actuator. In such an implementation environment, some simplifying assumptions become difficult to satisfy in practice. The most important of these is non-negligible delay introduced by contention on shared communication and sometimes computation resources.

A modern design flow must account for this constrained environment and simultaneously handle the vastly increased number of control applications and the ensuing complexity. Cars from 2013, for instance, ship with around 100 million lines of code [139], roughly 5 times more than the LINUX kernel in June 2015 [105]. Notwithstanding that this comparison may be misleading because car software is made up of components that probably include the LINUX kernel, cars have come a long way. “Even low-end cars now have 30 to 50 ECUs embedded in the body, doors, dash, roof, trunk, seats, and just about anywhere else … [In 1981,] GM was … executing about 50 000 lines of code across its entire domestic passenger car production.” [42]. The formerly hand-crafted code that runs on these ECUs is mostly auto-generated today by graphical frameworks that model the signal flow. This creates an additional layer of abstraction that must be reasoned about.

To deal with the growing complexity, different abstractions are used to analyze individual subsystems. A control engineer would typically start by considering the block diagram of a control problem. An example of such a diagram which helps to focus on the relations between

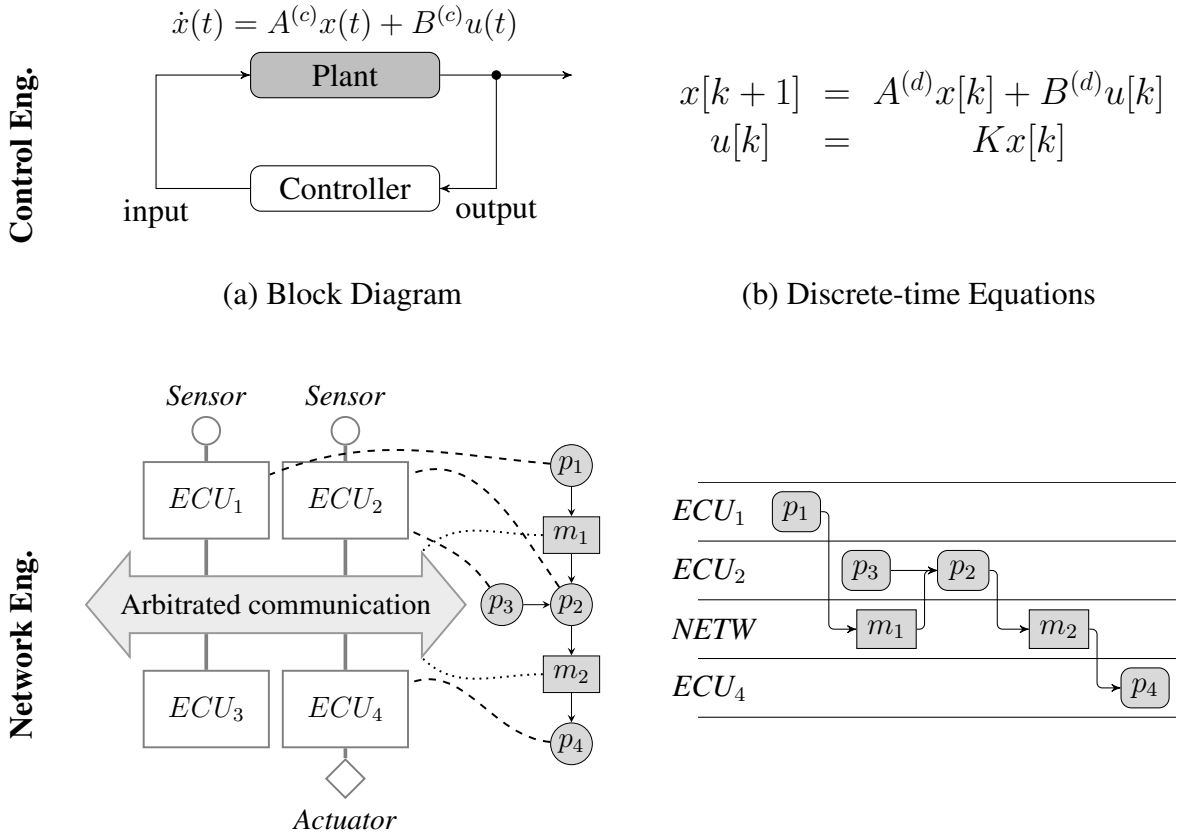
subsystems, is shown in Fig. 1.1(a). In this perspective, the engineer first models the so-called *plant*, consisting of the physical process to be controlled and an actuator. The plant is described in time domain by a standardized ODE system, the so-called *state space representation*. While there are alternatives like transfer functions, this form has advantages when analyzing systems with multiple in- and outputs or with nonlinear behavior, justifying its increasing general popularity. More importantly for the work at hand, this representation makes reasoning about timing analysis and delay issues more natural. To obtain such a mathematical representation, the plant is usually analyzed according to fundamental mechanical and electrical principles in bottom-up fashion. As an alternative, semi-automated or guided system identification [118] procedures are available that yield sufficiently accurate results in many circumstances.

Once a suitable plant description is available, the engineer deals with equations for control law design and discrete-time transformation as in Fig. 1.1(b). These operations can be performed in any order as long as sampling period and delay remain small compared to the change rate of the plant. During control law design, engineers search for a mapping, often a gain matrix K, between system state and input signal. This mapping, the control law, defines an action for each possible state. To find it, many techniques have been established, guaranteeing various properties for different system classes. The Linear Quadratic Gaussian (LQG) framework [11], for instance, minimizes a quadratic cost function using only a gain matrix and no optimization at runtime. To transform the continuous system to its discrete version, a fixed delay and sampling period are utilized as discussed earlier in this section. During this phase of control application design, the underlying hardware platform is hence typically taken into account according to this interface specifying period and maximum delay.

The network engineer, on the other hand, considers the dependency chain formed by the computer processes and messages that implement the corresponding control application shown in Fig. 1.1(c). He or she distributes the elements of this chain on a network of ECUs and communication links or buses. Once the computation cost of a process is established, it corresponds to a specific time requirement depending on the power of the selected ECU. Similarly, the size of a message implies the time it needs to be transmitted over a certain communication bus. With this knowledge, each process / message chain distribution corresponds to a timing diagram as in Fig. 1.1(d). Such a diagram details how the message timing and in particular how the end-to-end or sensor-to-actuator delay behaves. Several frameworks have been devised to find analytic bounds for the timings in such a network. Real-time Calculus (RTC) [178], for instance, applies a mathematical convolution at each processing element to a stream with given arrival behavior and obtains worst-case delays and backlog in this way, among other properties.

While distributing the elements of control applications in the network, the dependency of the processes, the location of sensors and actuators, as well as the timing specifications must be observed. This is a large set of complex conditions, but it may still leave significant flexibility. If the network is not fixed yet, one may be interested in its topology and its components. Even on a fixed network, process distribution, priority assignment, and other parameterizations remain complicated, yet relevant questions. Searching for an optimal network according to metrics like price, volume, or extensibility given such a set of constraints is commonly referred to as Design Space Exploration (DSE). There are many DSE techniques, relying on methods like integer programming, boolean satisfiability, or evolutionary algorithms [71].

In most network design endeavors, the control application requirements are considered as



(a) Block Diagram

(b) Discrete-time Equations

(c) Processes and messages on a network

(d) Timing Analysis

Figure 1.1: *Different views on digital control design.*

period / deadline specification. This specification matches the period / maximum delay description that abstracts the hardware platform during control design. There are several issues with this interface. Often, a very low sampling period is selected along with an ideal zero delay to achieve great performance during simulation. Sampling too often wastes computation and communication resources, however. More importantly, zero delay cannot be obtained in practice. A lengthy and costly test phase is then required to analyze the real behavior and guarantee the safety of a system. If the specification is selected to be more achievable, it may forgo control performance without leading to a real benefit on the network side.

Instead of a design process that is driven by either network or control requirements, the *Cyber-physical System (CPS)* paradigm aims for an integration of both sides and ideally a *co-design of control law and hardware platform.* Even in such a general setting, the interface itself may hide optimal solutions from the design process, however. This is the case when specifying only period and maximum delay. As the average is typically smaller than the worst-case delay in complex networks, operating a control application with shorter deadline and some timeouts often leads to improved control performance.

Since a fully holistic co-design with one model that includes detailed network and plant

behavior together remains infeasible, all interfaces must strike a balance between design speed and abstraction level. The computing power for off-line analysis has grown tremendously, however, bringing tools within reach that were infeasible only a decade ago. One such tool that is well-suited for the analysis of worst-case behavior is model-checking [12] as commonly applied in the area of digital circuit design.

In the first part of this thesis, we investigate how model-checking can support the co-design goals of the CPS paradigm. For this purpose, we mainly deal with an extension of the aforementioned RTC timing analysis framework that is based on model-checking, the Event Count Automaton (ECA) framework. Originally devised to analyze buffer requirements and delay under state-dependent scheduling behavior, it is also well-suited to treat sporadic deadline misses. On the control engineering side, we analyze the resulting performance on platforms whose behavior is guaranteed via ECA evaluation. As a further step, we then look for strategies that specifically rely on this interface of behavior guarantees. These approaches provide a tighter integration than state-of-the-art interfaces and can hence often improve the resulting control performance in a co-design setting.

1.2 Active Cell Balancing (ACB) design

Another challenge for CPS design are digital actuation schemes. In contrast to the timing-based interaction described in Section 1.1, these schemes may require special care even for physical processes that evolve much slower than any potential network delay. As a case where standard analysis is not sufficient, this part discusses charge transfer circuits for balancing battery packs at high frequency.

The background of these circuits is the dramatic proliferation of Electrical Energy Storage (EES) and the various applications it enables that occurred in the post-millennial years. Laptop sales now exceed those of desktop computers; smartphones and tablets have their own success stories. At the same time, there is a growing interest in Electric Vehicles (EVs) as well as residential EES. This interest is part of a general desire for clean energy and transportation arguably motivated by peak oil discussion, high fuel prices, and environmental concern. An EV may completely rely on electrical energy for zero local emissions or use it to improve the combustion engine's efficiency. Residential EES mainly aims to smoothen the non-homogeneous output of renewable power sources and to increase independence from the regional power grid.

While some applications favor other technologies like super-capacitors or fuel cells, most store electrical energy in Lithium-Ion (Li-Ion) battery cells because of their high energy density. When the electrodes of such a rechargeable cell are connected, an electrochemical reaction causes an electrical current to flow. The voltage that Li-Ion cells can achieve during this discharge process is around $4\,\mathrm{V}$, limited by the cell chemistry. To form the basis for a wide range of applications, many cells are hence typically connected in series and in parallel. While cells in parallel connection inherently reach an equilibrium and can be considered as electrical unit, the charging or discharging of serially connected cells must stop as soon as the first cell reaches its limit. Imbalances between these cells, caused by differences in capacity and internal resistance from production or by non-homogeneous cooling, hence reduce the effective capacity of the pack.

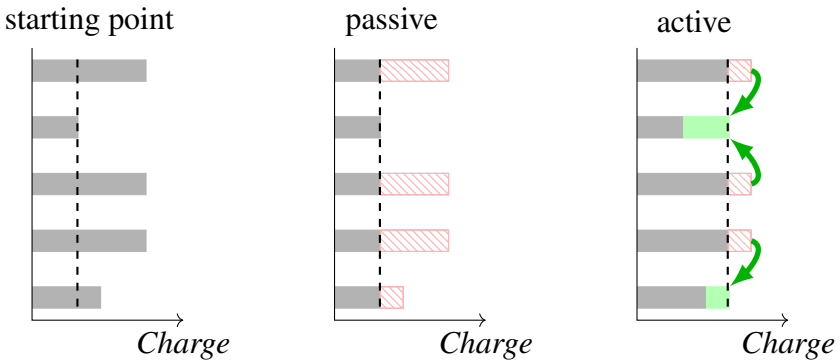


Figure 1.2: *Variations in cell characteristics lead to imbalances over time, leaving some cells with excess charge above the current pack level (dashed). Passive balancing, which is the current standard because it can easily be implemented, simply discards local excess charge (hatched). Active balancing techniques instead transfer the surplus and thus achieve higher pack charge levels.*

For applications that require high voltage, and thus many cells in series, imbalances are currently alleviated in two ways. First, cells with similar properties are clustered at production time to minimize deviations in capacity and internal resistance, e.g., by cell weight and volume. The deviations that are potentially avoided in this way exceed 5% [161]. This is not sufficient, however, since the properties of identical cells evolve differently over time, even under lab conditions [19]. Second, excess energy in individual cells is therefore dissipated using switchable resistors [82]. This technique, referred to as *passive balancing*, is easy to implement, but clearly not energy-efficient.

Alternatively, the excess charge can also be transferred to other cells. Fig. 1.2 illustrates schematically how this approach, referred to as *Active Cell Balancing (ACB)*, increases the usable energy of a battery pack and consequently its performance. While ACB can be implemented in numerous ways [32], the implementations can be grouped by the transfer mechanism they rely on. This mechanism is usually determined by the main circuit component or components that perform the charge transfer. For instance, capacitors can be connected in parallel to cells and charge or discharge them in this way. Such parallel connections lead to rather inefficient charge transfer, however. Alternatively, suitable shunting at runtime can create additional rest periods for cells with less charge. This approach balances the pack under load, but it leads to voltage fluctuation if not controlled and requires switches in the main series connection of the battery pack. Having switches there affects the efficiency during standard operation, however, which is usually not acceptable. The most efficient ACB implementations are built around inductors or transformers and actuated with switching signals in the kilohertz range.

To obtain a well-performing ACB application, several decisions, simulations, and optimizations are necessary. The following design flow summarizes the overall process.

1. Select transfer mechanism and define topology requirements.

2. Synthesize suitable circuit architecture including transistor switching scheme.

3. Determine size requirements and choose circuit components accordingly.

4. Design control strategy.

The first stage begins with the selection of the transfer mechanism. While there are several options, as just discussed, this thesis focuses on the class of inductor-based architectures for efficiency reasons. One then defines the topology of the circuit. This is mainly about the kind of transfers that the circuit shall be able to perform. Transferring only between adjacent cells may be sufficient in some cases; others may require transfers to non-adjacent or multiple cells. Another popular design has one cell always transfer to or from the whole pack. Further desirable properties include modularity and concurrency. Modularity helps when additional cells are added to a pack. Performing transfers concurrently, on the other hand, vastly reduces balancing time. If shorter balancing times are not required, concurrency can also lead to lower currents and consequently to higher efficiency in most cases.

In the second stage, a circuit architecture is synthesized according to the specification from the first stage. For this purpose, a network of transistors is created to connect transferring components, like inductors, to cells. The transistors are then switched according to a scheme that defines their configuration in several recurring phases. This switching realizes the actual charge transfer. While this stage only deals with qualitative behavior and component parameters do not play a role yet, switching scheme and circuit architecture must be analyzed together for potential issues like short circuits or undesired cell discharging. Although the architecture can be designed by hand and verified in a standard circuit simulator, dedicated tools for faster verification and even automated synthesis are also emerging [123]. As an alternative to synthesizing, it is also often possible to utilize an existing architecture from the literature, such as the inductor-based circuit for transfers between adjacent cells from [103].

The third stage deals with component selection or design in the circuit architecture obtained so far. The numerous, not necessarily identical transistors and most importantly the inductor determine under which scenario a circuit performs best. In the fourth stage, control strategies for the actuation at runtime are devised. Depending on charge distribution in the pack and available time, different current rates and transfer sequences may be beneficial. To justify decisions in these two stages, the system behavior must now be treated quantitatively. General purpose circuit simulators may be used for ACB evaluation, but they have drawbacks like low flexibility and long computation times, among other things. It is hard to implement sophisticated strategies in a circuit simulator because state information is hard to aggregate and many mathematical operations are not readily available. Similarly, many efficient optimization techniques are impossible because they require access on the equation level. Even simulation may become unacceptably slow for larger systems or longer time frames. The high-frequency switching that actuates the charge transfers leads to correspondingly small steps in the simulator. A balancing strategy, on the other hand, may have to be evaluated over several hours and therefore require millions of steps. In this area, switching details are hence often abstracted away, e.g., by averaging. While this leads to simple formulations, it also creates a modeling gap with additional relative errors of several percent points. On the other hand, faster simulation and mathematical programming techniques are also possible in a lossless fashion by directly looking at fundamental electrical principles, like Kirchhoff's laws. This provides more insight and direct access to equations of the transfer dynamics.

Even though this work contains contributions to the second stage, its focus is on the third

and fourth stage. It develops quantitative models that apply to the large class of inductor-based charge transfer circuits, pursuing two main goals. The first is accurate, but fast simulation of ACB. The second is optimization of ACB using mathematical programming. Although more complex than in Section 1.1, a transformation to discrete-time helps to identify actuation interfaces that formalize the long-term system behavior. The contributions to both goals build on this interim result as these interfaces lead to further important reformulations.

1.3 Contributions and organization

After introducing the background information from CPS co-design and ACB design in the previous sections, this section discusses the results of the thesis at hand. After listing contributions, it explains the organization of the remaining chapters, detailing in particular how they relate to previous publications of the author.

The main contributions of this work are summarized as follows.

- Integration of the Event Count Automaton (ECA) timing analysis framework into Cyber-physical System (CPS) design, creating an interface that allows direct verification of control performance for distributed linear systems from within the timing analysis.
- Development of a fault-tolerant control strategy design that uses the information about the implementation platform from ECA analysis in a Linear Matrix Inequality (LMI) formulation to yield an improved design under limited network resources.
- Addition of Fixed-Priority Non-preemptive Scheduling (FPNS) to the ECA framework, handling issues with priority inversion and the ensuing message blocking.
- Investigation and advancement of models for charge transfer dynamics, proposing a total of 3 models that are no longer anchored to the natural switching phases in the microsecond range and can hence be used for accurate, but rapid long-term simulation (speedup of 5 orders of magnitude compared to standard simulation) or mathematical programming.
- Development of a PYTHON library with C++ back end that implements the aforementioned transfer models used, for instance, in the distributed balancing co-simulator of the author's research group.
- Application of the models for optimal design of inductors for charge transfer architectures using Geometric Programming (GP), formalization of high-level actuation interfaces as well as design and evaluation of efficient charge routing strategies.

This thesis is organized in 7 chapters. The current chapter gives an overview of the environments it aims to improve, CPS co-design with a focus on delay effects as well as ACB design with switching actuation. By describing the current situation and challenges, it provides context for the contributions of the work. The contributions themselves are explained in greater detail in the remainder of the thesis which is structured as follows.

Chapter 2 provides background information on CPS analysis. It summarizes the standard approach for describing physical processes in control applications which we require for delay

analysis, the so-called state space representation. This representation exists in continuous-time and discrete-time form. The latter arises when a digital device senses or actuates a physical process, turning the overall setup into a sampled-data system. After describing the transformation from continuous- to discrete-time form, the chapter goes over the standard properties and performance measures for control applications. As we are interested not only in messages that arrive with delay, we then treat switched systems which are necessary to model messages that do not arrive at all. The chapter ends with an explanation of standard properties or variations thereof which are compatible with formal languages for model-checking. This content is included as reference; the contributions from the following chapters build upon this foundation in many cases.

Chapter 3 introduces the Event Count Automaton (ECA) framework, an approach for timing analysis of communication and computation networks. It first describes the two automaton types this framework employs, the service and the arrival ECA, both formally and with examples. The chapter then shows how multiple ECAs can be combined to form a network by connecting them using in- and output buffers. Modeling a hardware platform, such an ECA network can then be analyzed with model checking tools for various properties, like maximum delay. Although other tools are also suitable, this thesis relies exclusively on SAL [54] for model-checking. To give an impression of ECA analysis in practice, the chapter thus includes some code snippets from the Scheme-like, functional programming language utilized in SAL. Following the basic introduction of the ECA framework, an extension of its semantics is then shown to model FPNS in this paradigm. Unlike for time-triggered scheduling protocols that directly map to the slots of a discrete-time automaton like an ECA, a model for FPNS is not straightforward. Since the non-interruptible transmissions can begin not only at slot transitions but also within slots, the resulting communication behavior may depend on intra-slot arrival times that cannot be tracked using discrete-time automata. The modeling approach described here treats the bus in a conservative way to capture problematic behavior like priority inversion that a straightforward implementation would miss. The FPNS model and the corresponding extension of the ECA semantics have been originally presented in [92].

Chapter 4 describes two applications of the ECA framework in the CPS setting. Both applications utilize the fact that the worst-case delay in complex hardware platforms only rarely occurs. Since feedback control systems are inherently robust to a certain number of communication faults, they may not require the hardware over-provisioning a worst-case design entails. For this reason, the applications build on an interface that allows individual messages to miss their deadlines according to specified amounts or patterns. In the first application, this interface is used to implement given control applications with certain performance requirements on a hardware platform. To that end, the performance requirements are analyzed to find the tolerable patterns of deadline misses in a form that model-checking tools can treat. These patterns are subsequently verified on a hardware platform modeled by ECAs. This approach typically leads to specification-conform designs with higher bus utilization and hence reduced costs. The second application aims to improve control performance given a certain hardware platform. For this purpose, a fault-tolerant control strategy design is proposed that builds upon the same ECA-verified interface. Using a LMI formulation solved via mathematical programming, the approach finds a controller that takes into account the recent communication faults at runtime to generate an improved input signal. This design fulfills exponential stability, a common control

performance requirement, by construction. A description of these applications has appeared in [91] and [90], respectively.

Chapter 5 discusses quantitative models for inductor-based Active Cell Balancing (ACB). It begins by presenting a selection of circuit architectures from this class of charge transfer circuits with switching actuation and explains the fundamental operating principles. These circuits charge an inductor from one cell, then change the configuration of their routing switches and discharge the inductor into another cell. Each configuration transition leads to a non-differentiable change in the transfer dynamics which is challenging for standard simulation techniques and unwieldy for actuation strategy design. A numerical solver for such dynamics commonly operates with an adaptive step size approach, aiming to make only few, large steps as long as an associated error estimate fulfills its specification. The non-differentiable transitions force such solvers to make multiple steps for each configuration change. This is slow because switching occurs in the microsecond range while balancing operations may last for several hours and hence require millions of steps over hours of computation time. Many models that are computationally cheaper, on the other hand, are also simple, ignoring transistor switching effects, the evolution of voltages, or the parasitic reactions in the involved battery cells. Nevertheless, many improvements are possible, both towards accurate yet fast ACB simulation and more natural actuation interfaces. The improvements developed here build on an equivalent circuit modeling approach that abstracts nonlinear circuit components and treats each switch configuration, or phase, as individual ODE. These ODEs have closed-form solutions under virtually all parameterizations and can thus be converted to a recurrence relation, similar to the discrete-time form of the state space representation in digital control.

This discrete recurrence relation is still too slow for large-scale simulations but it enables high-level actuation interfaces and subsequently two rapid simulation models, each with its own advantages. The first, most precise, and most versatile model uses error control techniques to take large steps in the discrete recurrence relation. In this way, it remains compatible with various common battery models and application interfaces. The second approach achieves even faster simulation times by solving the recurrence relation in closed form, assuming an actuation interface that leaves switching periods constant for many iterations. One charge transfer architecture with detailed switching scheme and the equivalent circuit approach has been published in [88]. The first simulation model has been submitted to IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems in January 2016. The second model forms an integral part of the co-simulator published in [166].

Chapter 6 builds on the developed transfer models to evaluate and improve the performance of ACB operation in three aspects: inductor design, local operating current, and routing strategies. To that end, it describes a transfer model that caters specifically to optimization and accepts some mathematical transformations with accuracy loss in unlikely operating regions. It then details a GP that can be used to design an optimal inductor for an ACB circuit. Although this program also deals with the operating current, the computation is too involved to be performed at runtime. The chapter hence goes on to discuss a closed form solution for the optimal current and a best-case Linear Programming (LP) formulation for the charge routing problem. The former is now accessible to virtually any computing device, the latter can be efficiently solved on contemporary personal computers. The LP result is a reference value, indicating what a routing strategy can achieve, and a set of transfers that attain this value assuming ideal

voltages in all participating cells. As heuristic routing strategies without concurrent transfers almost achieve this value already, we then investigate the main constraints from the LP to find a strategy that parallelizes more naturally and remains suitable for implementation on embedded devices. Several heuristic routing strategies have been introduced in the papers that make up Chapter 5. Inductor optimization has been presented in [89].

In Chapter 7, we summarize the results and conclude this work. We reevaluate the proposed techniques in a bigger picture and suggest conditions to double-check before resorting to involved CPS design or ACB over simpler alternatives. Future research endeavors are motivated by pointing out potential and risk of the presented interfaces in the DSE context, the options for even more accurate charge transfer models, and rarely explored benefits of ACB.

1.4 List of publications and awards

The contributions to Cyber-physical System (CPS) co-design and the Event Count Automaton (ECA) framework from this thesis have appeared in the following publications:

- Matthias Kauer, Sebastian Steinhorst, Dip Goswami, Reinhard Schneider, Martin Lukasiewycz, Samarjit Chakraborty. *"Formal Verification of Distributed Controllers using Time-Stamped Event Count Automata"*. In: Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC 2013).

- Matthias Kauer, Sebastian Steinhorst, Reinhard Schneider, Martin Lukasiewycz, Samarjit Chakraborty. *"Automata-Theoretic Modeling of Fixed-Priority Non-Preemptive Scheduling for Formal Timing Verification"*. In: Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC 2014).

- Matthias Kauer, Damoon Soudbakhsh, Dip Goswami, Anuradha M. Annaswamy, Samarjit Chakraborty. *"Fault-Tolerant Control Synthesis and Verification of Distributed Embedded Systems"*. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2014).

The contributions to architectures, models, and optimization for Active Cell Balancing (ACB) have appeared in the following publications:

- Matthias Kauer, Swaminathan Narayanaswamy, Sebastian Steinhorst, Martin Lukasiewycz, Samarjit Chakraborty, Lars Hedrich. *"Modular System-Level Architecture for Concurrent Cell Balancing"*. In: Proceedings of the 50th Design Automation Conference (DAC 2013).

- Matthias Kauer, Swaminathan Narayanaswamy, Sebastian Steinhorst, Martin Lukasiewycz, Samarjit Chakraborty. *"Many-to-Many Active Cell Balancing Strategy Design"*. In: Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC 2015).

- Matthias Kauer, Swaminathan Narayanaswamy, Martin Lukasiewycz, Sebastian Steinhorst, Samarjit Chakraborty. *"Inductor Optimization for Active Cell Balancing using*

Geometric Programming". In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2015).

- Sebastian Steinhorst, Matthias Kauer, Arne Meeuw, Swaminathan Narayanaswamy, Martin Lukasiewycz, Samarjit Chakraborty. *"Cyber-Physical Co-Simulation Framework for Smart Cells in Scalable Battery Packs"*. To appear in: ACM Transactions on Design Automation of Electronic Systems (TODAES).

- Matthias Kauer, Swaminathan Narayanaswamy, Sebastian Steinhorst, Samarjit Chakraborty. *"Rapid Analysis of Active Cell Balancing Circuits"*. Under Submission.

The following publications are related to the topic of this thesis, but not a direct part hereof:

- Martin Lukasiewycz, Sebastian Steinhorst, Florian Sagstetter, Wanli Chang, Peter Waszecki, Matthias Kauer, Samarjit Chakraborty. *"Cyber-Physical Systems Design for Electric Vehicles"*. In: Proceedings of the 15th Euromicro Conference on Digital System Design (DSD 2012).

- Dip Goswami, Martin Lukasiewycz, Matthias Kauer, Sebastian Steinhorst, Alejandro Masrur, Samarjit Chakraborty, S Ramesh. *"Model-Based Development and Verification of Control Software for Electric Vehicles"*. In: Proceedings of the 50th Design Automation Conference (DAC 2013).

- Martin Lukasiewycz, Sebastian Steinhorst, Sidharta Andalam, Florian Sagstetter, Peter Waszecki, Wanli Chang, Matthias Kauer, Philipp Mundhenk, Suhaib A. Fahmy, Shreejith Shanker, Samarjit Chakraborty. *"System Architecture and Software Design for Electric Vehicles"*. In: Proceedings of the 50th Design Automation Conference (DAC 2013).

- Peter Waszecki, Matthias Kauer, Martin Lukasiewycz, Samarjit Chakraborty *"Implicit Intermittent Fault Detection in Distributed Systems"*. In: Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC 2014).

- Swaminathan Narayanaswamy, Sebastian Steinhorst, Martin Lukasiewycz, Matthias Kauer, Samarjit Chakraborty. *"Optimal Dimensioning of Active Cell Balancing Architectures"*. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2014).

- Sebastian Steinhorst, Martin Lukasiewycz, Swaminathan Narayanaswamy, Matthias Kauer, Samarjit Chakraborty. *"Smart Cells for Embedded Battery Management"*. In: Proceedings of the 2nd International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA 2014).

- Sebastian Steinhorst, Zili Shao, Samarjit Chakraborty, Matthias Kauer, Shuai Li, Martin Lukasiewycz, Swaminathan Narayanaswamy, Muhammad Usman Rafique, Qixin Wang. *"Distributed Reconfigurable Battery System Management Architectures"*. In: Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC 2016).

- Swaminathan Narayanaswamy, Matthias Kauer, Sebastian Steinhorst, Martin Lukasiewycz, Samarjit Chakraborty. *"Modular Active Charge Balancing for Scalable Battery Packs"*. Under Submission.

In addition, the work performed for this thesis has led to the following awards.

- *HiPEAC Paper Award, 2013*: Recognition for European contributions at conferences where Europe is not strongly represented.
- *A. Richard Newton Young Student Fellow Program, 2013*: Travel grant for ≈ 50 students at Design Automation Conference in Austin, TX
- *Global Young Scientists Summit, 2014*: Discussion opportunity with eminent technology leaders
- *ACM SIGDA CADathlon 2014 at International Conference On Computer Aided Design (ICCAD)*: ECA modeling paper [92] was selected as functional verification problem.

2 Cyber-physical System (CPS) Analysis

Cyber-physical Systems (CPSs) are a symbiosis of a dynamical system that is implemented on and thus interacts with a digital platform. The textbook example for this is the inverted pendulum experiment where a rod of a certain length has to be kept upright by moving its base in the plane. The strategy for these movements is programmed into a computer or a small microcontroller, depending on the setup, and together, they form a larger system that is referred to as CPS.

Depending on the focus of the analysis, such systems are also referred to as digital control systems or networked control systems. The former are the earliest variant and deal with the issues of periodic sampling as well as fixed delay; the latter add a fixed network into the analysis which is typically considered random in nature. The CPS paradigm expands on this, ultimately aiming for a co-design of network, or platform, and control system.

Although they have an individual focus, the underlying theory of the aforementioned approaches overlaps significantly. *This chapter introduces the major results from literature that the later parts of the thesis rely on.* It first presents general forms of dynamical systems (Section 2.1) and the communication hardware (Section 2.2) that is commonly used to implement them in a distributed fashion. As soon as the digital impact becomes significant for its evolution, a system must be considered as sampled-data system (Section 2.3), typically in discrete time. Next, the chapter goes over the most relevant properties of a control system (Section 2.4). In order to account for message losses, in addition to delay, we then explore switched systems (Section 2.5). These systems evolve differently depending on discrete events like a successful message arrival. Unlike for switching actuation as discussed in Chapter 5, switching is considered adversarial here and assumed to arrive in the worst pattern. Finally, we explore which of the desirable control system properties can be applied to switched systems and expressed in a form that is compatible with model checking (Section 2.6). In this way, control system analysis can be combined with the model checking-based timing analysis from Chapter 3; this combination is the topic of Chapter 4.

2.1 Continuous dynamical systems

A system is first of all an entity formed by interacting components that are in some way separated from the rest of the world. Across its boundaries, the system interacts with its surroundings via established input and output signals. For dynamical systems described in mathematical terms, one very general formulation is

$$\dot{x} = f(x, u). \tag{2.1}$$

Here, x is the state vector that a control engineer attempts to influence via input vector u. If f is not benign in some sense, very few statements can be made about such systems in general (see textbook [93] for reference). Nevertheless, this form already makes an assumption of causality. The system is only affected by its input. The input device, on the other hand, is not affected by the system unless explicitly modeled.

Since the most general form is hard to reason about, researchers have mostly dealt with less general versions. The most popular paradigm is that of Linear Time-invariant (LTI) systems. While LTI systems cannot describe all applications, they do cover the basic properties for virtually all systems. When evaluating a control system, the first property that comes to mind is usually *stability* (see Section 2.4). Informally speaking, one wonders whether the system can possibly drive itself and accelerate out of control. This question is directly linked to an equilibrium or working point x_e, u_e with $f(x_e, u_e) = 0$ that can be used for linearization. By defining new coordinates around this point $\Delta x = x - x_e$, the system can be described linearly and becomes much easier to analyze. With continuous system matrices

$$A^{(c)} = \left[\frac{\partial f}{\partial x}\right]_{x_e,u_e} \qquad B^{(c)} = \left[\frac{\partial f}{\partial u}\right]_{x_e,u_e}$$

and re-setting $x = \Delta x$, i.e., considering only the new coordinate system, one arrives at a general form for LTI systems.

$$\begin{aligned} \dot{x}(t) &= A^{(c)}x(t) + B^{(c)}u(t) \\ y(t) &= C^{(c)}x(t) \end{aligned} \tag{2.2}$$

Besides linearization, this so-called *state space representation* expands upon (2.1) by introducing output vector y. y can model that, in general, not all states, i.e., not all elements of x are measurable from the outside. $A^{(c)}$, $B^{(c)}$, $C^{(c)}$ are called continuous system, input, and output matrix, respectively. They fully characterize the system behavior. The superscript $^{(c)}$ distinguishes the continuous from the discrete version of the system, as examined in Section 2.3. It is often clear from the context which system description is meant and we can then simply refer to the matrices as A, B, C.

Converting to an LTI system is highly desirable since they come with a large number of formal results. For the context of this work, the most relevant properties that become verifiable and even designable are stability as well as quadratic costs for penalizing tracking error and input energy. Please refer to Section 2.4 for an overview.

The accuracy of a linear model rapidly decreases outside the working point, however. Caution is thus required when evaluating properties without taking the distance from the working point into account.

In addition, the form in Eq. (2.2) is only adequate to describe systems with negligible delay and continuous actuation. Contemporary systems are mostly implemented on distributed digital platforms, however (Section 2.2). This rules out continuous actuation because it would require continuously recomputing the input signal and hence needs infinite computing resources. In addition, contention on both computing and communication resources leads to waiting times. Whereas (2.2) assumes that a new input $u(t)$ can be applied the moment $x(t)$ occurs in the system, there is always a delay in reality. Consequently, we discuss in Section 2.3 the sampled-data system modeling which includes sampling periods and possibly non-negligible delay.

In certain cases, an additive disturbance term $w(t)$ is included in the LTI state space representation from Eq. (2.2). This term is supposed to take both random noise inside the system as well as modeling errors into account. Many design approaches ignore these disturbance terms and rely on the inherent robustness of the control loop.

$$\begin{aligned} \dot{x}(t) =& A^{(c)}x(t) + B^{(c)}u(t) + w(t) \\ y(t) =& C^{(c)}x(t) \end{aligned} \tag{2.3}$$

Typically, disturbance $w(t)$ is assumed to be normally distributed, written $w(t) \sim \mathcal{N}(0, \Sigma_c)$. This choice is mostly driven by the desire to make algebraic techniques manageable. Questions about average performance in particular benefit from the normal distribution. When considering the worst case, the normal distribution is rarely chosen because its infinite support leads to infinitely bad performance bounds.

2.2 Communication hardware

Distributed implementations of feedback control applications are common in many safety-critical domains like automotive and avionics. In this work, we consider setups where a feedback control application is implemented on multiple ECUs that communicate over a shared bus. As shown in Fig. 2.1, the sensing devices and the actuators are connected to different ECUs and the control algorithm uses feedback signals that are transferred via the shared bus system. While being transmitted over such a bus, feedback signals often get delayed due to contention by other messages, thereby resulting in a *sensor-to-actuator* delay.

Delay in a feedback control system has tremendous influence on its performance. It would thus be desirable to route all communication between the various ECUs directly. Such a fully connected mesh requires many wires, however, that are then under-utilized on average. The implied cost and extra weight almost always rule out this approach. Currently, the automotive industry and many others hence favor bus topologies. These require the least amount of cabling but also lead to more contention.

To keep a shared bus usable, access must be arbitrated in some way. The major paradigms are currently time-triggered and event-triggered arbitration.

Time-triggered arbitration The highest amount of predictability and reliability can be achieved by granting each application access to the bus in periodic fashion. To make this work, a fixed period of time is divided into slots. This period, referred to as *superframe* in many architectures, is typically in the low millisecond range. Each slot is then assigned to an application or

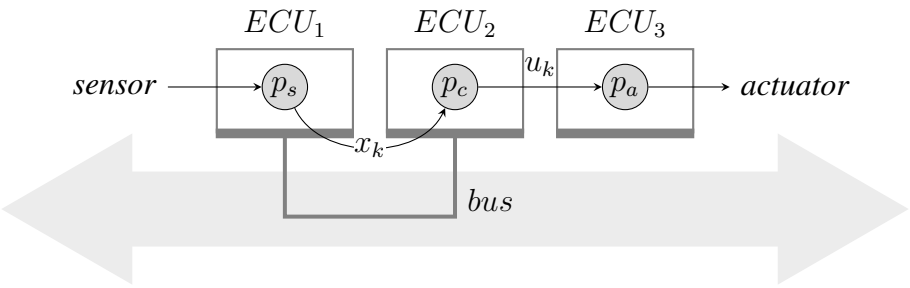


Figure 2.1: *In distributed control applications, sensor task p_s, controller task p_c, and actuator task p_a are implemented on separate ECUs and communicate over a shared bus. While this allows for execution on specialized processors and often reduces the number of ECUs required overall, it also introduces delays.*

ECU. As time evolves, the participants access the bus one after another during their designated time slots. Once an entire cycle is completed, the procedure immediately starts again. Consider textbook [97] for reference.

One downside of time-triggered arbitration is the low average utilization. If an application does not have information to distribute during its assigned slot, the transmission opportunity is wasted. The other participants become aware of such an unused slot too late to use it. In most cases, they should also not abandon their own periodic schedule. The total bandwidth that must be available for time-triggered arbitration is therefore larger than the actually required bandwidth.

To achieve high data rates on time-triggered platforms, an accurate synchronization of the clocks in all participants is usually required. This synchronization is challenging but necessary to keep the network functional. Without synchronization, the transmissions of different ECUs would overlap once their clocks drift apart enough and neither of the messages could arrive. This is catastrophic for the applications relying on them because there is usually no fall-back mechanism.

Event-triggered arbitration Instead of distributing access to a bus in slots, participants can also try to access the bus according to their needs. If the bus utilization is above a certain level, this almost certainly leads to collisions. Event-triggered schemes manage these collisions in different ways. In IEEE 802.11, the protocol behind wireless local area network, nodes sense the bus before attempting a transmission. If two nodes still start sending at the same time, they back off and restart their attempt after waiting a random period. While this leads to high throughput on average, such schemes are problematic for CPS implementation because they produce highly indeterministic waiting times.

The Controller Area Network (CAN) bus has a more suitable arbitration scheme, also referred to as Fixed-Priority Non-preemptive Scheduling (FPNS). All participants are assigned a priority that simultaneously serves as their address. Every message begins with this ID. The physical layer is implemented in such a way that a higher priority ID can overwrite others should they attempt transmitting at the same time. This keeps the arbitration overhead low and leads to predictable behavior. Most importantly, the high-priority devices enjoy very low delay. The

CAN arbitration also means, however, that a high-priority process can keep others from sending for a long time. In other words, low-priority processes can be "starved".

In order to guarantee a timely arrival of all important messages on a CAN bus, several measures are usually taken. The average bus utilization is kept low in the hope that the reserve thus created is sufficient to handle the worst case access patterns. Over-dimensioning the hardware in this way undoubtedly leads to higher costs, however. In addition, the system undergoes extensive test runs and, possibly, subsequent re-design with the knowledge obtained during the tests.

Compared to time-triggered arbitration, event-triggered schemes lead to higher bandwidth utilization and are consequently cheaper. On the other hand, time-triggered schemes inherently guarantee a certain arrival time for all participants. The associated delay is thus predictable and easy to work with. While guaranteed, the delay may still be larger, however, than in the worst case of the event-triggered version, to the dismay of the control engineer. It is a misconception that a real-time guarantee means that a system is fast in some sense. On the contrary, achieving such guarantees usually slows the system down significantly. Albert summarizes the trade-off between event-triggered and time-triggered systems as follows. "The main advantage of event-triggered systems is their ability to [quickly] react to asynchronous external events which are not known in advance [...]. Thus, they show a better real-time performance in comparison with time-triggered systems. In addition, event-triggered systems possess a higher flexibility and allow in many cases the adaptation to the actual demand without a redesign of the complete system." [2]

In the past, the low costs and the challenges associated with time-triggered arbitration have led to wide-spread deployment of event-triggered communication buses, like the CAN bus. The associated issues discussed above were not of immediate concern since the utilization on most CAN buses was rather low in the beginning. Issues like varying wait times producing jitter and starvation have surfaced only later with higher load on the bus. Today, new bus systems like FLEXRAY with a mix of time- and event-triggered segments and a higher bandwidth have arrived. Notwithstanding, CAN continues to be cost-effective in many environments and will also need to be supported as legacy platform for many years to come.

Design process The design processes for time-triggered and event-triggered systems have orthogonal advantages. Designing with priorities in mind is easy at first, but not compositional. The system can only be analyzed with full knowledge about how many messages each participant might send. Slot-based designs require more thought right away, whereas event-triggered schemes "just work" at the outset. Only in larger systems where slots can be distributed to individual teams do time-triggered approaches really provide an advantage. In these circumstances, one engineer can arrange his application according to the assigned slots without worrying about contention from others.

In terms of redesign on the other hand, event-triggered schemes may have the upper hand. There, adding another application to the bus may be as simple as assigning a new priority to it if sufficient bandwidth is still available. In time-triggered schemes, such situations may require a redistribution of slots when the available ones are not in a suitable pattern for the application to be included.

2.3 Sampled-data systems

Contemporary control systems are implemented on digital hardware. The perspective of the control engineer on such a system is shown in Fig. 2.2. It forms a contrast to the network and computer focused perspective from Fig. 2.1. The plant is now in focus whereas before it was only implicit on the boundaries. The controller interacts with it through sensor and actuator. State $x[k]$ is still transmitted over the network. The implementation details of the communication platform are abstracted away, however.

CPS timing and delay Fig. 2.3 shows the timing diagram that the CPS faces on the hardware. The control engineer calls the delay between the sensor reading and the actuator applying a new signal based on that information the *sensor-to-actuator delay* d. The network engineer can now implement the platform in an interrupt-driven fashion, meaning p_a triggers as soon as the required information becomes available. In this case the control engineer's delay d becomes the *varying transmission delay* τ, i.e., $d = \tau$, neglecting the brief time required for p_a itself. The control design must then take varying delay into account which is quite complicated.

Alternatively, the network engineer can consider control delay d to be a deadline for his transmissions. For this purpose, he stores the signal in a buffer until p_a triggers at the predefined instant. The platform and the controller are then sized such that $\tau \leq d$ which significantly simplifies the analysis. This is the approach taken in this work.

This work is particularly interested in the effects of delay. In order to study the phenomenons associated herewith, we limit ourselves to LTI systems of the form (2.2). Whether d, the delay in the control realm is constant or not, it can be taken into account there as follows.

$$\begin{aligned} \dot{x}(t) =& A^{(c)}x(t) + B^{(c)}u(t-d) + w(t) \\ y(t) =& C^{(c)}x(t) \end{aligned} \tag{2.4}$$

This still assumes an ongoing re-computation of the input signal, another issue that we discuss next.

Periodic sampling With a delayed communication pattern as in Fig. 2.3, it is unlikely that computation occurs in an ongoing fashion as Eq. (2.4) assumes. In fact, continuous computation would require infinite resources. Instead, one selects a sampling period h and interacts with the

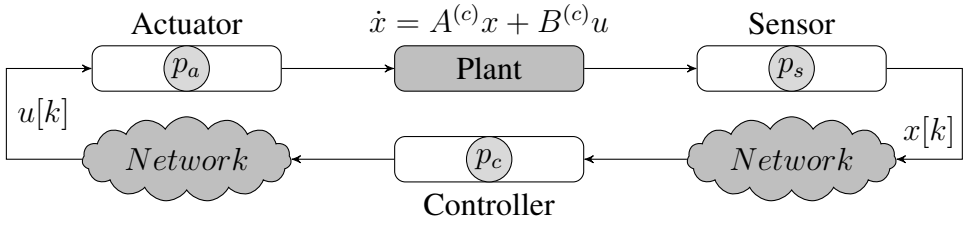


Figure 2.2: *The control engineering perspective of the distributed application from Fig. 2.1 has sensor and actuator directly interacting with the plant while the controller computes input signals after receiving sensor readings over a network.*

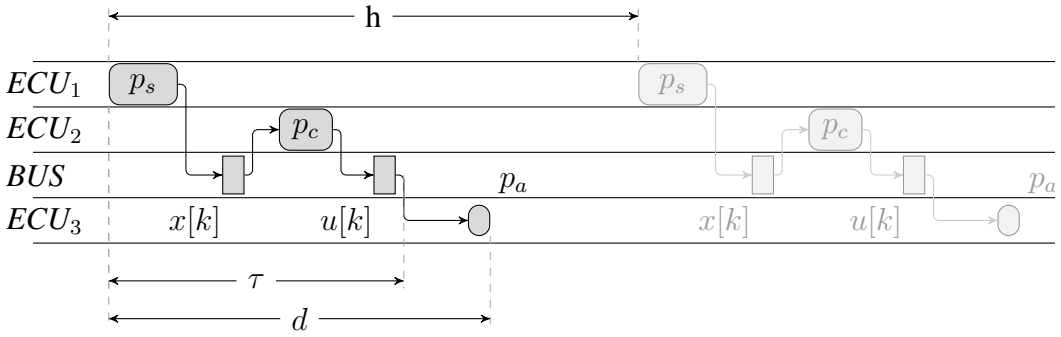


Figure 2.3: *The timing diagram of a distributed controller shows that sensor task p_s, controller task p_c, and actuator task p_a all require computation time on their respective resources. Communication occurs over a shared bus; waiting time dominates actual transmission times there. In a* buffered *implementation, p_a is triggered periodically with an offset d that serves as deadline. Input signal $u[k]$ must be present at that moment for the system to proceed without failure, i.e., the varying* sensor-to-buffer *delay τ must not exceed deadline d.*

system periodically. In other words, each sensing, computation and actuation task triggers a fixed amount of time since the last occurrence has elapsed. This corresponds to the situation depicted by Fig. 2.3.

The periodicity introduced by sampling period h implicitly defines time instants $\{t_k\}_{k\in\mathbb{N}}$ as $t_k = kh$. Input signal u is calculated once, applied with delay d and then constant for a full period. As long as h remains small, it is also reasonable to assume that disturbance w remains constant within a sampling period.

$$\begin{aligned} w(t) &= w[k], t \in \big[t_k, t_{k+1}\big) \\ u(t) &= u[k], t \in \big[t_k + d, t_{k+1} + d\big) \end{aligned} \tag{2.5}$$

Notice how control delay d is reflected in the definition of input u. Åstrom and Wittenmark's textbook [10] contains more details on the systems that this sampling induces.

Actuation pattern (2.5) is often called Zero-Order Hold (ZOH). In the purest form of ZOH the system is actuated by an input signal as soon as it arrives. The actuator then holds that signal until new instructions arrive. ZOH is a very basic, interrupt-driven scheme. If the delay is assumed to be constant and message drops are excluded, ZOH looks indeed identical to (2.5) from the perspective of a control engineer.

Conversion to sampled-data system We can now insert the sampling input signals and calculate the system evolution using ODE (2.4). This involves repeatedly solving the ODE and may not be efficient on embedded devices. The standard approach instead transforms the ODE into a recurrence equation custom fit to the sampling period. State-of-the-art ODE solvers with adaptive step size are extremely fast for long-term LTI simulations because they can calculate many sampling periods at once. The ongoing input signals can be computed faster through the simpler matrix multiplications of the specific recurrence, however. More importantly, standard ODE techniques can no longer be applied if delay forms a part of the system model. Searching

for a solution then involves functional differential equations [152] which are significantly more challenging.

We detail the transformation from continuous ODE to discrete recurrence relation in the following. Using the periodically constant input pattern from (2.5), we first obtain the discrete-time representation for $x[k] = x(t_k)$ of the undisturbed system ($w(t) = 0$). This requires only standard ODE solution techniques like the variation of constants method.

$$\begin{aligned} x(t_k) =& e^{A^{(c)}h} x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{A^{(c)}\nu} B^{(c)} u(\nu)\,\mathrm{d}\nu \\ =& e^{A^{(c)}h} x(t_{k-1}) + \int_0^d e^{A^{(c)}\nu}\,\mathrm{d}\nu \quad B^{(c)} u(t_{k-1}) \\ &+ \int_d^h e^{A^{(c)}\nu}\,\mathrm{d}\nu \quad B^{(c)} u(t_k) \end{aligned} \tag{2.6}$$

This shows that the dependency of $x[k]$ on its predecessor $x[k-1]$ is described by a matrix $A^{(d)} = e^{A^{(c)}h}$, the discrete system matrix. If $A^{(c)}$ is invertible, we can utilize the Taylor series representation of the matrix exponential to efficiently calculate discrete input matrix $B^{(d)}$.

$$\begin{aligned} \int_0^d \exp(A\nu)\,\mathrm{d}\nu = \int_0^d \sum_{l=0}^{\infty} \frac{1}{l!} A^l \nu^l\,\mathrm{d}\nu &= \sum_{l=0}^{\infty} \frac{1}{(l+1)!} A^l d^{l+1} \\ &= A^{-1} \cdot (\exp(Ad) - \mathbf{1}) \end{aligned} \tag{2.7}$$

Here, $\mathbf{1}$ refers to the identity matrix. Overall, we can thus reformulate Eq. (2.6) to

$$x[k+1] = A^{(d)} x[k] + B_1^{(d)} u[k-1] + B_0^{(d)} u[k]. \tag{2.8}$$

In this difference equation for the system dynamics, the new matrices $A^{(d)}$, $B_1^{(d)}$, $B_0^{(d)}$ containing its parameters are obtained as follows.

$$\begin{aligned} A^{(d)} &= e^{A^{(c)} \cdot h} \\ B_1^{(d)} &= \int_0^d e^{A^{(c)} \cdot \nu}\,\mathrm{d}\nu B^{(c)} = (A^{(c)})^{-1} \left[e^{A^{(c)}d} - \mathbf{1}\right] B^{(c)} \\ B_0^{(d)} &= (A^{(c)})^{-1} \left[e^{A^{(c)}(h-d)} - \mathbf{1}\right] B^{(c)} \end{aligned} \tag{2.9}$$

As long as delay d and sampling period h are constant, these matrices can be computed off-line. Since their computation is otherwise quite expensive, employing buffers to prevent timing variations is typically justified.

If there is a state feedback control law, $u[k] = Kx[k]$, Eq. (2.8) is often re-written to a state space form with *augmented state* $\begin{bmatrix} x[k] & x[k-1] \end{bmatrix}'$. Substituting the control law into Eq. (2.8) yields

$$x[k+1] = A^{(d)} x[k] + B_1^{(d)} K x[k-1] + B_0^{(d)} K x[k]. \tag{2.10}$$

This can be written as state space representation for the augmented state.

$$\begin{bmatrix} x[k+1] \\ x[k] \end{bmatrix} = \begin{bmatrix} A^{(d)} + B_0^{(d)} K & B_1^{(d)} K \\ \mathbf{1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x[k] \\ x[k-1] \end{bmatrix} \tag{2.11}$$

Here, $\mathbf{1}$ and $\mathbf{0}$ refer to identity and zero matrices of appropriate size. One must be cautious not to re-write the system in this form before the control law is fixed. Otherwise, the simulation may inadvertently propagate information from the current state to the next state in a non-causal fashion.

While the single matrix form facilitates simulation and analysis in certain cases, it also shows that the delay has effectively doubled the dimension of the state space. This increase is usually not critical for most controller designs and especially simulations. It constitutes a real harm for discretization approaches, however, where computational effort increases exponentially with the dimension of the state space.

Transforming disturbance parameters to discrete time We have established the discrete-time dynamics of the undisturbed system in (2.8). If the original system included a disturbance term, like (2.3), the characterization of this disturbance must be transformed as well. We thus derive a discrete-time form of covariance matrix $\Sigma^{(c)}$ by integrating the constant disturbance term along with the original dynamics that we had solved starting from Eq. (2.6).

$$x[k+1] = A^{(d)} x[k] + B_1^{(d)} u[k-1] + B_0^{(d)} u[k] + \underbrace{\int_{t_k}^{t_{k+1}} e^{A^{(c)}\nu} w(\nu)\, \mathrm{d}\nu}_{w[k]} \tag{2.12}$$

From here, we can calculate $\Sigma^{(d)} = \mathbf{E}\left[w[k]w[k]'\right]$ using $w[k] = \int_0^h e^{A^{(c)}\nu} w(\nu)\, \mathrm{d}\nu$.

$$\begin{aligned} \Sigma^{(d)} =& \mathbf{E}\Big[\int_0^h \int_0^h e^{A^{(c)}\nu} w(\nu) w'(\alpha) e^{A^{(c)\prime}\alpha}\, \mathrm{d}\alpha\, \mathrm{d}\nu\Big] \\ =& \int_0^h \int_0^h e^{A^{(c)}\nu} \mathbf{E}\left[w(\nu)w'(\alpha)\right] e^{A^{(c)\prime}\alpha}\, \mathrm{d}\alpha\, \mathrm{d}\nu \\ =& \int_0^h e^{A^{(c)}\nu} \Sigma^{(c)} e^{A^{(c)\prime}\nu}\, \mathrm{d}\nu \end{aligned} \tag{2.13}$$

In the last step we have used that the correlation over time is zero for Gaussian noise. This yields $\mathbf{E}\left[w(\nu)w^T(\alpha)\right] = \Sigma^{(c)}\delta(\nu - \alpha)$. Numerically, we have two options to calculate $\Sigma^{(d)}$. We can employ the method described in [179]. There, van Loan suggests computing

$$\exp\Big(\begin{bmatrix} -A^{(c)} & \Sigma^{(c)} \\ 0 & A^{(c)\prime} \end{bmatrix} h\Big) = \begin{bmatrix} (A^{(d)})^{-1} & (A^{(d)})^{-1}\Sigma^{(d)} \\ 0 & A^{(d)\prime} \end{bmatrix}$$

and subsequently obtaining $\Sigma^{(d)}$ as a combination of block matrices:

$$\Sigma^{(d)} = \left(A^{(d)\prime}\right)'\left((A^{(d)})^{-1}\Sigma^{(d)}\right) \tag{2.14}$$

Alternatively, we can assume $e^{A^{(c)}\nu} \approx \mathbf{1}$ for $\nu \in [0, h]$ and obtain

$$\Sigma^{(d)} \approx \int_0^h \mathbf{1}\Sigma^{(c)}\mathbf{1}'\nu \,\mathrm{d}\nu = \Sigma^{(c)} \cdot h. \tag{2.15}$$

2.4 Quality of Control

The motivation behind the efforts of control systems analysis is to find a suitable, or even an optimal (with respect to some metric) input signal u. Whether this search is conducted for the general form (2.1) or the simpler LTI versions, requirements and desirable properties must be discussed beforehand.

Stability

Stability is the quintessential requirement of a closed-loop dynamic system. In [67], Friedland defines it as "the ability of the system to operate under a variety of conditions without 'self-destructing' ".

Given its fundamental importance, many definitions have been introduced and a variety of techniques have been developed to test for stability. The most basic stability definition asks whether the system will return to its equilibrium after a disturbance. Lyapunov's definition is the most commonly accepted version nowadays, arguably because it applies to a very broad class of systems. It is reproduced in the following with the assumption that the system has been adjusted such that $x = 0$ is an equilibrium. Consider Khalil's textbook [93] for further background information.

Definition 2.1 (Stability & asymptotic stability)**.** *Consider an autonomous dynamical system*

$$\dot{x}(t) = f(x(t)) \qquad\qquad x(0) = x_0$$

with $f : D \to \mathbb{R}^n$, a locally Lipschitz map from a domain $D \subseteq \mathbb{R}^n$ into $\mathbb{R}^n$. The equilibrium point $x_e = 0$ where $f(x_e) = 0$ is stable if for each $\epsilon > 0$, there is $\delta = \delta(\epsilon) > 0$ such that

$$\|x(0)\| < \delta \Rightarrow \|x(t)\| < \epsilon \quad \forall t \geq 0.$$

If the equilibrium is not stable, it is called unstable.

In addition, $x = 0$ is called asymptotically stable if it is stable and δ can be chosen such that

$$\|x(0)\| < \delta \Rightarrow \lim_{t\to\infty} x(t) = 0.$$

Definition 2.1 asks whether we can find a δ-region such that we are guaranteed not to leave a previously selected ϵ-neighborhood. In other words, given a small region, can we find a possibly even smaller starting region such that we stay in the specified region for all times.

In the nonlinear case, proving stability usually involves finding a Lyapunov function. Such a function models the energy level of the system in some sense. The idea is that as long as energy decreases over time the system must eventually reach its equilibrium. Not every system

has a formulation for its energy readily available, however. In other cases, the available version may not produce the desired result. Lyapunov has thus proven that using alternative energy descriptions can be used to prove stability as well. This is detailed in the following Theorem, reproduced from [93].

Theorem 2.2 (Lyapunov function)**.** *Let $x = 0$ be an equilibrium point for an autonomous system*

$$\dot{x}(t) = f(x(t)) \qquad\qquad x(0) = x_0$$

with $f : D \to \mathbb{R}^n$, a locally Lipschitz map from a domain $D \subseteq \mathbb{R}^n$ into $\mathbb{R}^n$. Let further $V : D \to \mathbb{R}$ be a continuously differentiable function with the following properties.

- *$V(0) = 0$ and $V(x) > 0$ for $x \in D \setminus \{0\}$*
- *$\dot{V}(x) \leq 0$ in D*

Then $x = 0$ is stable. Moreover, if $\dot{V}(x) < 0$ in $D \setminus \{0\}$, then $x = 0$ is asymptotically stable.

Lyapunov functions are a tremendously helpful tool. Their main disadvantage is that it often requires expert knowledge to find them. Recipes for designing them are only available for certain simpler system classes.

Evaluating stability for linear systems like (2.2) is possible without Lyapunov functions altogether. They have usually been designed around a working point. This working point subsequently becomes their origin and the main equilibrium we are interested in. One very common test examines the eigenvalues of system matrix $A^{(c)}$. Since the unforced version of the system $\dot{x} = Ax$ has solutions of the form $x(t) = e^{At}x_0$, we can learn about stability from the eigenvalues of A. If the real parts of all eigenvalues are strictly negative, e^{At} tends to zero and we have an *asymptotically stable* system. If there is a strictly positive eigenvalue, the system is *unstable*. With zero real parts, the situation is more complicated. Such systems are referred to as *stable, but not asymptotically stable* and tend to exhibit ongoing oscillations. Fig. 2.4 explains these concepts using abstract one-dimensional systems.

A discrete-time linear system (2.8) can be examined similarly. There, however, we must check that the eigenvalues are within the unit circle of the complex plane for a stable system.

Other popular tests that check for stability from the literature are the following. Root locus method and Nyquist stability criterion are both graphical approaches that helped jump start the field when manual computation was the only option. They are still used in robust and adaptive control settings. Similarly, the Bode diagram can be used for graphic stability checks in the frequency domain and the Hurwitz criterion is a quicker alternative to computing the eigenvalues of the system matrix that evaluates smaller determinants instead.

Exponential stability

Stability is the most basic requirement for control systems, but it is often not sufficient. We would also like to avoid ongoing oscillation and very slow convergence. Exponential stability hence expands upon this concept by specifying a convergence speed in terms of an exponential decay. The general definition for this requirement is as follows.

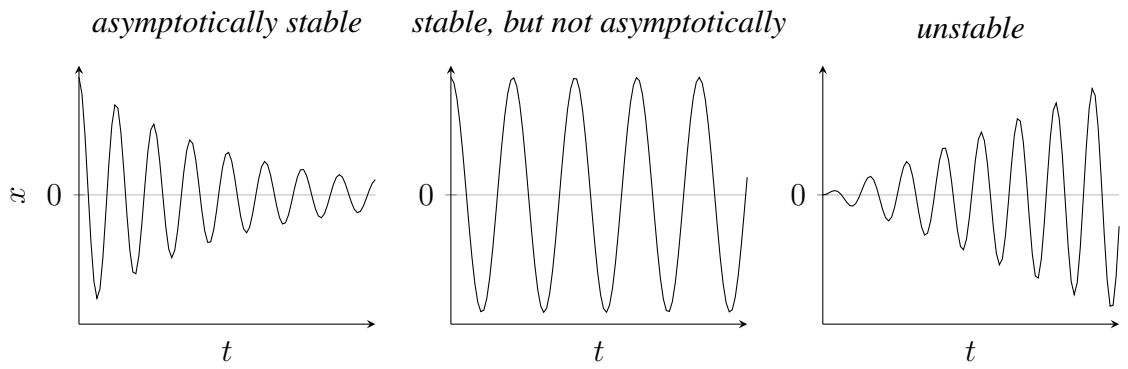


Figure 2.4: *A control system, whether one-dimensional like here or not, behaves in one of three stability patterns. An asymptotically stable system eliminates disturbances by itself and shrinks toward its working point, $x = 0$ in general. An unstable system increases errors and grows toward infinity. In between, there is the case where deviations lead to future deviations of identical size.*

Definition 2.3 (Exponential stability)**.** *Consider an autonomous dynamical system*

$$\dot{x}(t) = f(x(t)) \qquad x(0) = x_0$$

with $f : D \to \mathbb{R}^n$, a locally Lipschitz map from a domain $D \subseteq \mathbb{R}^n$ into $\mathbb{R}^n$. We call equilibrium $x_e = 0$ where $f(x_e) = 0$ exponentially stable if it is asymptotically stable and there exist α, β, δ such that

$$\|x_0\| \leq \delta \Rightarrow \|x(t)\| \leq \alpha \|x_0\| e^{-\beta t} \quad \forall t \geq 0.$$

For continuous LTI systems where solutions have the form $x(t) = e^{At}x_0$, an exponential decay requires the (necessarily negative) eigenvalues to be smaller than a certain threshold value.

For discrete-time systems exponential stability can be defined with regards to the time steps. In this context, exponential stability *ExpStab*(l, ϵ) as model checking property in the way Weiss and Alur present it (see Section 2.6 or [185]) requires

$$\frac{\|x_{k+l}\|}{\|x_k\|} < \epsilon, \tag{2.16}$$

where $\|.\|$ denotes the 2-norm. In other words, for a plant to be considered exponentially stable, any error must be reduced by at least a factor of ϵ in l sampling periods, i.e., a time of $\Delta t = l \cdot h$ where h is the sampling period.

Quadratic costs

Designs for stability leave several parameters free for further design. Optimal control goes beyond merely ensuring stability and instead uses an additional performance criterion.

One way to measure the performance of a system, the so-called Quality of Control (QoC), is looking at the associated continuous linear quadratic control costs.

$$J_C^{(c)} = \int_0^T x'(t) Q^{(c)} x(t) + u'(t) R^{(c)} u(t) \, \mathrm{d}t \tag{2.17}$$

Here, cost matrices $Q^{(c)} \succeq 0$, $R^{(c)} \succ 0$ are used to weight the importance of each individual state and input signal. For the cost formulation to be meaningful the involved matrices Q, R must be positive semi-definite or definite, respectively. These terms are explained in Definition 2.4 after this paragraph. Otherwise, Formulation (2.17) would encourage making errors or wasting input energy. To understand this, consider a cost of $J^{(c)} = \int_0^T -u'(t)u(t)$. Clearly, minimizing these costs will maximize the input energy. Note that most solution techniques do not even work in these uninteresting circumstances.

Definition 2.4 (Positive definiteness)**.** *A symmetric matrix $X \in \mathbb{R}^{n\times n}$ is called positive definite if $z'Xz > 0$ for all $z \in \mathbb{R}^n \setminus \{0\}$. It is called positive semi-definite if $z'Xz \geq 0$ for all $z \in \mathbb{R}^n$.*

For two symmetric matrices $X, Y \in \mathbb{R}^{n\times n}$, X is larger than Y in the positive definite sense if $X - Y$ is positive definite. This is written

$$X \succ Y \Leftrightarrow X - Y \succ 0 \Leftrightarrow z'(X - Y)z > 0 \quad \forall z \in \mathbb{R}^n \setminus \{0\}$$

The comparison operator $\succeq$ for "greater equal" or "greater in the positive semi-definite sense" is defined analogously.

Remark 2.5 (Positive definiteness via eigenvalues)**.** *A symmetric matrix $X \in \mathbb{R}^{n\times n}$ is positive definite according to Definition 2.4 if and only if all its eigenvalues are positive.*

$$X \succ 0 \Leftrightarrow \lambda_i > 0 \quad \forall \lambda_i \in \lambda(X)$$

Similarly, X is positive semi-definite if its eigenvalues are non-negative.

$$X \succeq 0 \Leftrightarrow \lambda_i \geq 0 \quad \forall \lambda_i \in \lambda(X)$$

Many textbooks are concerned with the design of a gain matrix that provides optimal performance with respect to linear quadratic costs (2.17). In other words, they look for a matrix K such that actuating the system with $u = Kx$ minimizes $J_C^{(c)}$. This Linear Quadratic Gaussian (LQG) regulator can be calculated by solving a matrix Riccati equation, an off-line operation that scales adequately. The entire framework is detailed in [168], for instance. Unstable systems yield infinite costs over an infinite horizon and, figuratively speaking, the optimal designs are thus automatically stable. A numerical solution approach for the underlying Riccati equation is presented in [17]. The task is also part of the MATLAB control system toolbox.

In certain cases, it may not be possible to efficiently find an optimal gain or other criteria may take priority. This is the case for systems that experience communication interference, for instance. In such situations, the quadratic costs from (2.17) can still be used to evaluate and compare the control quality of different approaches.

Next, consider the discrete representation of the linear quadratic control costs from Eq. (2.17). It is common to write them as follows.

$$J_C^{(d)} = \sum_{k=0}^{N} x'[k]Q^{(d)}x[k] + u'[k-1]R_1^{(d)}u[k-1] + u'[k]R_0^{(d)}u[k] \tag{2.18}$$

The involved matrices can be transformed accurately with the approach presented in [9] and the matrix exponential manipulations from [179]. A simpler approach is to ignore second- and higher-order terms to obtain

$$Q^{(d)} = h \cdot Q^{(c)} \qquad R_1^{(d)} = d \cdot R^{(c)} \qquad R_0^{(d)} = (h-d) \cdot R^{(c)}.$$

It often makes sense to combine $R_1^{(d)}$ and $R_0^{(d)}$. Consider the following transformation of (2.18). It exploits that $u[k-1]$ is the input of the previous step and ignores the border cases that become irrelevant as soon as N, the number of steps to consider, becomes large.

$$J_C^{(d)} = \sum_{k=0}^{N} x'[k]Q^{(d)}x[k] + u'[k]\big(R_1^{(d)} + R_0^{(d)}\big)u[k] + u'[-1]R_1^{(d)}u[-1] - u'[N]R_1^{(d)}u[N]$$

The discrete costs are therefore often written with a single $R^{(d)} = R_1^{(d)} + R_0^{(d)}$.

$$J_C^{(d)} = \sum_{k=0}^{N} x'[k]Q^{(d)}x[k] + u'[k]R^{(d)}u[k] \tag{2.19}$$

The LQG framework continues to work with discrete dynamics. This means there is another gain matrix $K^{(d)}$ such that actuating with $u[k] = K^{(d)}x[k]$ minimizes $J_C^{(d)}$. Consider Bertsekas' textbook [21] for background information on how the involved recursion converges to the Discrete Algebraic Riccati Equation.

2.5 Linear switched systems

When physical systems depend on digital environments in any way, we have to design appropriate sampling and we will usually encounter some form of delay. This is taken into account by the sampled data systems in Section 2.3. Digital components additionally introduce switching, however. This can be voluntary, e.g., when scheduling computation tasks or involuntary, e.g., when messages do not arrive in time. These behaviors are analyzed from the control perspective in a context referred to as switched systems [114].

A switched system moves from one subsystem to another. The switches can be driven by the environment, considered as opponent or disturbance, or they can be voluntary if we move between specialized controllers. If all subsystems are linear, we end up with a linear switched system of the following form.

$$\dot{x} = A_\sigma x \tag{2.20}$$

Here, $\{A_\sigma\}_\sigma$ is a set of system matrices that represent the various subsystems. In this work, we will usually treat $\{A_o, A_c\}$ differentiating only between closed loop, i.e., standard, desirable conditions and open loop operation that occurs when we lose messages for some reason. If we have a controller K designed for feedback operation of a system described by matrix A and we decide to input nothing without new information, we obtain the subsystems

$$\dot{x} = \big(\underbrace{A + BK}_{A_c}\big)x \qquad \dot{x} = \big(\underbrace{A}_{A_o}\big)x. \tag{2.21}$$

The first question for all control systems is stability. When switching is considered, this is typically answered by finding a common Lyapunov function, i.e., a single Lyapunov function (see Theorem 2.2) that decreases for all systems. As long as the energy of all all subsystems decreases with respect to one common function, the overall system is also stable. This is not necessarily the case if all subsystems are stable but only with respect to individual Lyapunov functions. Branicky shows a counter-example for this in [29].

For linear switched systems we do not need to rely on expert intuition to find such a Lyapunov function. Theorem 2.6, taken from [126] and presented next, represents a computational tool for this search.

Theorem 2.6 (Common Quadratic Lyapunov Function)**.** *A switching system*

$$x[k+1] = A_i x[k], \quad i = 1, 2 \cdots N. \tag{2.22}$$

is exponentially stable with Common Quadratic Lyapunov Function (CQLF) $V(x) = x'Px$ *if there exists a matrix* $P \succ 0$ *solving the system of LMIs*

$$A_i' P A_i - P \prec 0, \quad i \in 1, 2 \cdots N \tag{2.23}$$

Here, $X \succ 0$ specifies that X has to be positive definite (Definition 2.4). LMIs are convex optimization problems [26]. As such, they can be efficiently solved.

2.6 Formally verifiable properties for switched systems

Traditionally, a feedback control application is designed to tolerate a maximum specified sensor-to-actuator delay and the underlying implementation architecture (ECUs and bus schedules) is chosen to meet such an end-to-end delay requirement.

Hence, the specified maximum sensor-to-actuator delay for which the controller has been designed acts as a *deadline* for the feedback control messages. Towards this, a control application is mapped onto an architecture only when *all* instances of control messages are guaranteed to meet their deadlines. Such a design approach where control performance requirements are mapped to *hard* timing constraints often turns out to be pessimistic and fails to exploit the inherent robustness of the feedback control loops.

Alur and Weiss have paved the way for an alternative approach. In [185] and [5], they formulate a number of common performance requirements for control systems in a way that formal verification tools understand. They treat discrete-time LTI systems without disturbance as in Eq. (2.8) and use finite Büchi automata as interface. We can verify specifications in this form by intersecting them with automata models for the underlying hardware platform. A text book like [12] details the well established algorithms that are required for this.

This section presents some of the Büchi automaton properties of Alur and Weiss. It then expands on them by grouping the numerous patterns in a coarser, but more manageable way, the (f,H)-firm deadlines. Furthermore, it adds thoughts on Linear Temporal Logic (LTL) versions, extending their use case to model checking tools that answer specifically whether a LTL formula is satisfied by a certain transition system.

Exponential stability requirement as Büchi automaton Alur and Weiss transform performance requirements as in Section 2.4 into Büchi automata. Their original motivation is on-line task scheduling. In other words, after their construction, they traverse many automata simultaneously and search for a path that appears in their intersection and therefore satisfies all the requirements. The motivation in this work is performance verification but the same construction is still useful. After encoding the performance requirements in the same way, we subsequently verify whether a certain architecture satisfies them.

Stability is the quintessential property for control systems. Nobody has formulated the basic variant in a way that a model checking tool could understand, however. Figuratively speaking, the issue is that stability cannot be decided with a finite view of the system. The stronger requirement of exponential stability is more amenable for formal verification. There, it is sufficient to select a certain period and collect all switching patterns the system can take that fulfill the decay requirement over that period.

Starting from (2.16)) and considering a switched system of the form (2.20), we obtain the following relation,

$$x_{k+l} = A_{\sigma_{k+l}} \cdots A_{\sigma_k} x_k \Rightarrow \frac{\|x_{k+l}\|}{\|x_k\|} \leq \left\| A_{\sigma_{k+l}} \cdots A_{\sigma_k} \right\| \tag{2.24}$$

In other words, the exponential stability requirement can be re-written to ([185])

$$\mathit{ExpStab}(l, \epsilon) = \left\{ \sigma_i \in \{o, c\}^\omega : \|A_{\sigma_{k+l}} \cdots A_{\sigma_{k+1}}\| < \epsilon \quad \forall k \in \mathbb{N} \right\}. \tag{2.25}$$

In model checking terms, this is a language of strings σ over the alphabet $\{o, c\}$. All strings correspond to switching patterns of A_o and A_c that ensure a possible tracking error in the system is reduced by at least a factor of ϵ over l sampling periods. [185] shows that this language is ω-regular. This means that it can be represented by a finite Non-determistic Büchi Automaton (NBA) and is consequently suitable for model checking.

Quadratic cost bound as Büchi automaton In addition to exponential stability, we may also be interested in the worst case performance of a switching system measured according to the quadratic costs from (2.19). In a switching system there is no input signal besides the switching. Costs for input energy must thus be imposed by penalizing the state in a suitable fashion. To reach a compatible formulation, we are also limited to finite horizon costs. Specifying an upper bound J over any horizon of K time steps, this yields the following switching patterns.

$$\mathit{Cost}(\{Q_i\}, K, J) = \left\{ \sigma_i \in \{o, c\}^\omega : \sum_{k=k_o}^{k_0+K} \sum_{i=1}^{m} x'[k] Q_i x'[k] < J \quad \forall k_0 \in \mathbb{N}, x_{k_0} \in D \right\} \tag{2.26}$$

Once more, [185] shows that this formulation is compatible with model checking.

(f,H)-firm deadlines Earlier parts of this section discuss how certain performance requirements for CPS correspond to testable patterns in switched systems. The computation of such sets of acceptable patterns can be done via brute-force search as in [185], but it becomes tedious to verify them pattern-by-pattern. To avoid this, we summarize many patterns in a combination of the following deadline constraint.

Definition 2.7 ((f,H)-firm deadline)**.** *A stream of control messages is said to fulfill the (f,H)-firm deadline τ if at least f out of any H consecutive messages meet their deadline. A system switching between two states $\{o, c\}$, open loop and closed loop, equivalently fulfills a (f,H)-firm specification if at least f out of any H consecutive steps are spent in closed-loop operation c.*

This definition interacts well with the moving window perspective taken in (2.25) and (2.26). The idea is that among all possible patterns, one can rule out all the unacceptable ones by a combination of (f, H)-firm deadlines. We will see that (f,H)-firm deadlines can be readily modeled using LTL. The hope is of course that checking for fewer such patterns is faster than evaluating all patterns that the presented brute-force NBA approach would yield. This is certainly true for tools that run their checks independently of each other. Section 4.1 has an example of how this can be applied.

Example: Switched system performance under (f,H)-firm specification The following motivational case study demonstrates the varying performance of a switched system over two similar (f,H)-firm deadline specifications. It discusses a setup made up of

$$A_c = \begin{bmatrix} 0.4000 & 0.0750 \\ -0.8250 & 0.3750 \end{bmatrix} \qquad A_o = \begin{bmatrix} 1.4300 & -0.3550 \\ 0.5330 & 1.5620 \end{bmatrix}. \tag{2.27}$$

The discrete-time arrangement evolves according to (2.22) with $i \in \{c, o\}$ switching between stable "closed-loop" and unstable "open-loop" mode.

Fig. 2.5 shows the difference in performance between architectures that fulfill various (f,H)-firm deadline requirements. Under a (5,5)-firm deadline, i.e., under ideal transmission conditions, the system settles quickly to its equilibrium after $k \approx 4$ steps. When 2 drops are allowed in a (3,5)-firm setting, the system performs noticeably worse. Allowing another drop in the (2,5)-firm specification more than doubles the settling time to $k \approx 11$ and lets the system move away from the equilibrium at first ($x_4 > x_0$!). This illustrates that some drops can be tolerated but once a threshold is reached control performance rapidly deteriorates.

Formulating properties in Linear Temporal Logic (LTL) Model checking frameworks usually cannot treat inputs in the form of (f,H)-firm deadlines directly. In order to interact with them, one has to resort to a language they understand. LTL is arguably the most common input interface for these tools.

The elementary operators of LTL, resembling other temporal logics, are introduced in [12] as follows.

- $\Diamond$ – "eventually" (at some point in the future), also expressed as **F** for "finally"
- $\Box$ – "always" (now and forever from now on), also expressed as **G** for "globally"

These operators are prepositions for LTL formulas that are constructed according to the following grammar.

$$\varphi ::= \text{true} \;\big|\; a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \neg\varphi \;\big|\; \mathrm{X}\,\varphi \;\big|\; \varphi_1 \,\mathrm{U}\, \varphi_2 \tag{2.28}$$

Here, the elementary formula elements are unary negation $\neg$, binary and $\wedge$, next operator X which delays checking for one time step and the until operator U. $\varphi_1 \,\mathrm{U}\, \varphi_2$ is true if and only

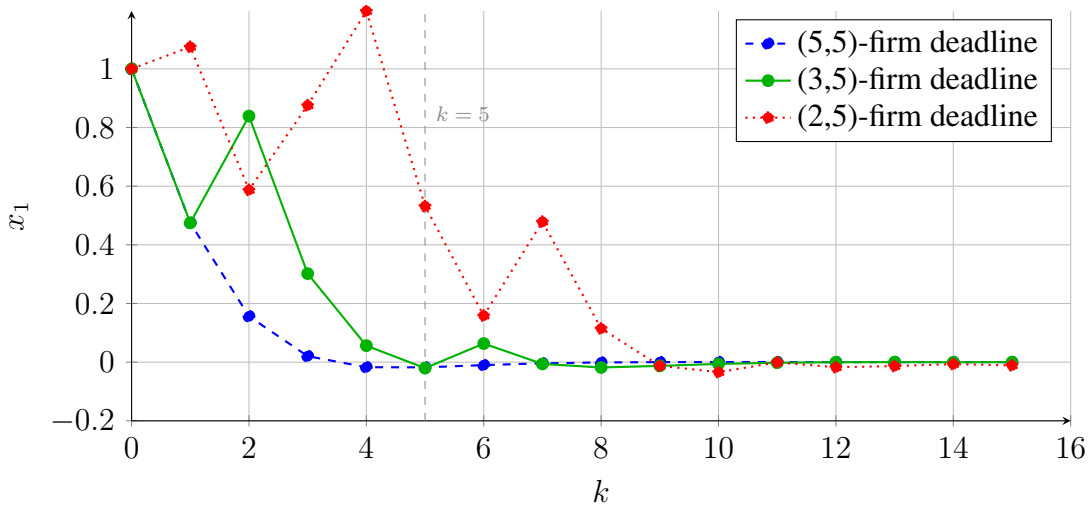


Figure 2.5: *Switched system performance typically deteriorates with less reliable communication.*

if φ_1 holds until φ_2 holds. Finally, $a \in AP$ refers to an atomic proposition which evaluates to true or false depending on the current state of the system. In our context, the set of atomic propositions is often given by

- $AP = \{o, c\}$ referring to a closed or open loop system, i.e., a successful or unsuccessful message transmission, or
- $AP = \{1, \ldots, N\}$ describing which indexed application is currently scheduled.

The elementary LTL operators from (2.28) can be extended via transformations with a handful of equally basic operators. The ones we use are the or operator $\vee$, implication $\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$, as well as weak until where $\varphi_1 \,\mathrm{W}\, \varphi_2 = (\varphi_1 \,\mathrm{U}\, \varphi_2) \vee \Box\varphi_1$ evaluates to true if φ_1 is true forever or holds until φ_2 holds.

A great help when formulating LTL properties is the patterns project [59], a collection of common LTL specifications. To express (f,H)-firm deadlines, as in Definition 2.7, we slightly modify the bounded existence pattern there. This yields LTL formula of the following form, allowing one failure over H cycles in this case.

$$\Box\Big\{(k = k_0) \Rightarrow \neg o \,\mathrm{W}\, \Big((k = k_0 + H) \vee \tag{2.29}$$

$$\big[o \wedge \mathrm{X}[\neg o \,\mathrm{W}(k = k_0 + H)]\big] \tag{2.30}$$

$$\Big)\Big\} \tag{2.31}$$

In this formula, $k = k_0$ and $k = k_0 + H$ are atomic propositions that check whether we are in a certain time step. o refers to being open loop mode because a transmission was unsuccessful. In the ECA framework that we introduce in Section 3.1, this can be evaluated with evaluation automata (see Fig. 3.5). The direct translation to the model checking tool SAL is in Listing 3.2 in Section 3.4.

The specification says that during the window $[k_0, k_0+H]$, o, meaning a failure, can occur at most once. This LTL formulation has to be checked for every possible k_0 of the time steps that run round-robin with maximum value H. H is the period that shall be observed. For instance, if the stream has a $(1, 3)$-firm deadline, then $H = 3$ has to be chosen. The amount of acceptable deadline misses can be varied by adding additional lines similar to (2.30). The idea behind this formulation is that we first require no failure ($\neg o$) until the end ($k = k_0 + H$) or until a failure o. If that failure occurs, however, $\neq o$ must then hold from the next step until the end. If there are multiple formulations of this kind, they can be checked sequentially.

3

Modeling Communication Platforms with Event Count Automata (ECAs)

ECAs, as introduced in [41], are a modeling language for timing analysis. This thesis uses them to describe how the hardware platform in a control system behaves. Chapter 4 presents how this behavior analysis interfaces with the control system properties from Chapter 2.

After introducing how an individual ECA can model a bus or processor by example and then formally (Section 3.1), this chapter describes how ECAs can be connected in networks (Section 3.2) to describe entire computation platforms. Given an ECA model, helper automata or other formulations can be used to verify various properties, like delay (Section 3.3). For this purpose, the ECA model has to be analyzed by a model checking tool. This role is filled by SAL [54] in this thesis (Section 3.4).

Next, we discuss how to model Fixed-Priority Non-preemptive Scheduling (FPNS) with discrete-time automata, like ECAs. Since the non-interruptible transmissions do not need to align with the time slots of an ECA, the scheduling behavior depends on intra-slot arrival times that cannot be tracked. As originally presented in [92], common models do not take this into account (Section 3.5). A non-deterministic model (Section 3.6) is necessary to perform conservative timing analysis, as demonstrated in the corresponding case study (Section 3.7).

The chapter ends with a discussion on related work (Section 3.8).

3.1 Event Count Automata (ECAs)

ECAs have originally been devised for stream processing applications. The stream processing abstraction comprises applications like audio/video or network packet processing, but also database access. A wide variety of devices, ranging from mobile phones, set-top boxes to network routers run such applications. In this formalism, applications are partitioned into several

tasks and subsequently mapped onto individual Processing Elements (PEs). A data stream then flows into the first PE where it is processed and forwarded to the next PE of the network. PEs are connected via buffers that store the partially processed data of the stream. Once a data element has gone through all PEs and their respective tasks, it is fully processed and leaves the network.

The challenge in these applications arises from the burst patterns that the streams exhibit. Determining the worst case delay of a stream or sufficient buffer sizes that prevent overflow can thus be a challenge. These are very natural questions during the design phase of processing hardware and frameworks like Network Calculus (NC) and, later, Real-time Calculus (RTC) have been developed specifically to answer them.

Both NC and RTC are functional frameworks in the sense that streams and PEs are described by algebraic functions. The analysis is performed purely through algebraic operations, typically convolutions of input stream and processor. While this makes them highly efficient, they face difficulty encoding state information. Special tweaks, as in [47] for Fixed-Priority Non-preemptive Scheduling (FPNS), are necessary to address these limitations and some properties can simply not be included without considerable pessimism.

ECAs aim to encode state information in a natural way while remaining as compatible with RTC as possible. [41] shows how an arrival curve, the function RTC uses to model inputs of a PE, can be converted to an ECA.

The reverse is not true in general. An ECA is more expressive than an algebraic function from the RTC framework, mainly because it can store state information. While significantly more expensive to evaluate as automaton, it remains useful because it can be combined with RTC. In this way, small but critical system parts may be modeled using ECAs while RTC describes the remaining network. This has been demonstrated in [145].

ECA example

An ECA describes event arrivals or occurrences. These may represent data coming from the environment or indicate the amount of processing available in a PE. A specific sequence of events is a string of integers. For example, in the case of an arrival ECA “201” denotes an arrival pattern with two messages arriving in the first interval, no messages in the second interval and one message in the third interval. In the case of a service ECA, it denotes the amount of data that is processed during the respective intervals. In total, an ECA models a set of strings containing all strings it can generate. This set is also referred to as its *language*.

The example ECA in Fig. 3.1 has a single count variable x. It accepts strings that have one event occurring in either state A or B. The following sequence is possible, for instance. We start with count variable $x = 0$ in state A. Now, zero or one event could occur. This is specified by the rate function $\rho(A) = [0, 1]$. Whether an event occurs or not, we can now take the only transition to B. In such a situation, transitions must be taken in the realm of ECA and we thus move to B. Assume that one event occurred while in state A. $x = 1$ therefore already. As specified by the guard $x = 1$ on transition (B, C), no further events occur and we move to C. There, the rate function $\rho(C) = [0, 0]$ specifies that no events can occur. We thus move back to A, resetting x. This movement yields the string ”100”.

Instead of occurring in state A as in the cycle just described, an event could also have

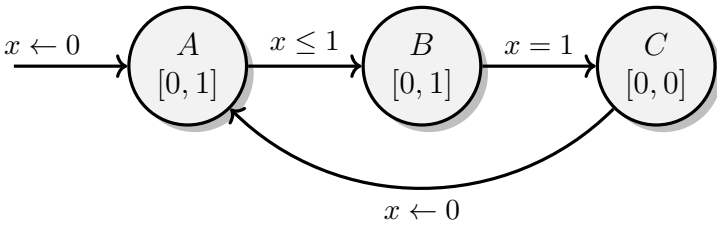


Figure 3.1: *Periodic with jitter arrival ECA ($p = 3, j = 2$)*

materialized in state B. This would yield the string "010". Combinations of the two strings constitute the overall language

$$\{100100\ldots, 100010\ldots, 010100\ldots, \ldots\}.$$

The ECA from Fig. 3.1 therefore models a process with period $p = 3$ and a jitter of $j = 2$. Note that the discrete nature of the ECA must assume that an event occurs at any time during its discrete time step. This means that an ECA that appears perfectly periodic already comes with a built-in jitter $j = 1$.

The next paragraphs present a more rigorous description of ECA semantics.

Formal ECA definition

ECAs track the number of events that take place in a unit interval of time. Dealing with such unit intervals, it is assumed that a suitable granularity of time has been fixed a priori. The arrival of events is tracked by *all count variables* associated with the automaton.

An ECA starts in a configuration (s_{in}, V_{in}) – an initial state and an initial valuation of all count variables. Transitions and, in particular, their individual guards determine when the ECA moves from state to state. During a transition – assumed to be instantaneous – some of the count variables can be reset to 0. In addition to guards on transitions, ECA movement can be directed by state invariant constraints. Altogether, an ECA is given by a tuple

$$\mathcal{A} = (S, s_{in}, X, V_{in}, \mathit{Inv}, \rho, \rightarrow)\,. \tag{3.1}$$

The components of this formulation are explained in the following.

- S is the set of states and $s_{in} \in S$ is the initial state.
- X is the set of count variables.
- V_{in} is the initial valuation of the count variables.
- $\mathit{Inv} : S \rightarrow \Phi(X)$ is the Invariant Constraint Function with

  $$\Phi(X) = x \leq c | x < c | x \geq c | x > c | \varphi_1 \wedge \varphi_2$$

  detailing the possible constraints. *Inv* assigns invariance constraints to the states.

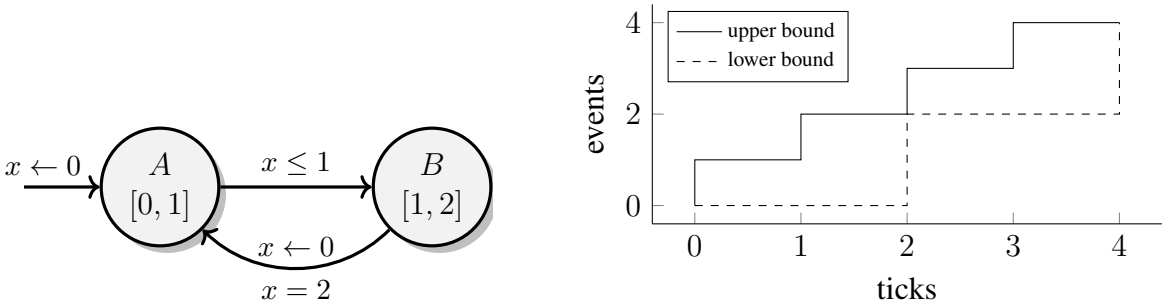


Figure 3.2: *Periodic with jitter arrival automaton ($p = 2, j = 2$) and corresponding upper and lower arrival curve.*

- $\rho : S \to \mathbb{N} \times \mathbb{N}$ is the rate function. Every state is assigned an interval

$$\rho(s) = [l, u]$$

  that specifies the possible rates of input arrival or service in that state.

- $\to \subseteq S \times \Phi(X) \times 2^X \times S$ is the transition relation. Each transition links two states, hence the association with $S \times S$, and it is equipped by a guard from $\Phi(X)$. It can further reset any subset of X to 0. These subsets make up 2^X, as usual.

The constraint variables c are integers; the rate intervals $[l, u]$ are integer intervals. For this reason, a count variable x can reach only integer values by design. Transitions are considered urgent, i.e., they have to be taken if possible. If no transition is possible, the automaton can also remain in its current state.

The *language* of ECAs contains *infinite* strings of integers that denote certain data arrival or processing patterns. A string $\sigma = n_1 n_2 \cdots \in [0, \rho_{\max}]^\omega$ is accepted if and only if the automaton can produce an infinite sequence $(s_0, V_0) \overset{n_1}{\Rightarrow} (s_1, V_1) \overset{n_2}{\Rightarrow} (s_2, V_2) \Rightarrow \ldots$.

In the case of the periodic with jitter automaton in Fig. 3.2, the states are given by $S = \{A, B\}$. The initial state is $s_{in} = A$ and the only count variable x is initialized to $x = 0$. There are no state invariants, the rate function yields $\rho(A) = [0, 1]$ and $\rho(B) = [1, 2]$. Finally, the transition relation is given by

$$\to = \Big\{ \big(A, B, x \leq 1, \emptyset\big) ; \big(B, A, x = 2, \{x\}\big) \Big\}.$$

The language of this ECA is represented by the upper and lower bound for arrivals depicted on the right in Fig. 3.2. From its initial state $s_{in} = A$, the guards are such that the ECA always returns to A in two steps. There may or may not be an event in A at first. In this case, the guard $x = 2$ forces two events to occur in B. The corresponding strings "11" and "02" obtained from this logic are then the building blocks of the overall language.

3.2 ECA networks

An ECA by itself only models the behavior of one component. For system-level results, we must connect multiple of them in the right way. The notion of ECA networks was introduced

for this purpose. These are essentially directed graphs where each vertex is associated with an ECA and every edge corresponds to a buffer. In each time step, the ECA nodes consume items from their input buffers and deposit items into their output buffers.

Formally, an ECA network is represented by a structure

$$\mathcal{N} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \{U_p\}_{p\in\mathcal{P}}, \mathcal{B}, B_{\max}, C, IN, OUT)\,. \tag{3.2}$$

This definition contains many elements that require further clarification.

- $\mathcal{P}$ is a finite set of nodes that the various ECAs $\mathcal{A}_p$ are associated with. A $\mathcal{P}$-indexed family such as $\{\mathcal{A}_p\}_{p\in\mathcal{P}}$ is often expressed as just $\{\mathcal{A}_p\}$.
- Each ECA is further associated with an update function U_p that defines from which buffers it consumes items and where it deposits them. They are described in more detail in Definition 3.1. Most update functions are trivial and the concept will become clear from examples. They play an integral role, however, in the modeling of FPNS for the ECA framework (Section 3.5).
- $\mathcal{B}$ is a finite set of buffers and $B_{\max}$ specifies their capacity. This maximum buffer capacity could be modeled for each buffer individually, but that is usually forgone for convenience reasons.
- C is the maximum number of items that any ECA in the network can handle in one step. Such a bound must be specified so we can guarantee that the state space behind the ECA that we need to explore remains finite.
- $\mathit{IN} : \mathcal{P} \to 2^{\mathcal{B}}$ and $\mathit{OUT} : \mathcal{P} \to 2^{\mathcal{B}}$ link the buffers to the ECA. For instance, $\mathit{IN}(PE) = B_1$ in Fig. 3.3. These functions are required to have the following properties.
  - $\mathit{IN}(p) \cup \mathit{OUT}(p) \neq \emptyset \quad \forall p \in \mathcal{P}$ – Each ECA $\mathcal{A}_p$ is connected to at least one buffer.
  - $\mathit{IN}(p) \cap \mathit{OUT}(p) = \emptyset \quad \forall p \in \mathcal{P}$ – ECAs do not deposit into their own input buffers.
  - $\mathit{IN}(p) \cap \mathit{IN}(q) = \emptyset = \mathit{OUT}(p) \cap \mathit{OUT}(q) \quad \forall p \neq q$ – Buffers are point to point and unshared.
  - $\forall B \in \mathcal{B}, \exists p : B \in \mathit{IN}(p) \vee B \in \mathit{OUT}(p)$ – All buffers are connected to at least one ECA.

Definition 3.1 (ECA update function)**.** *Let $\mathcal{A}_p$ be an ECA with $N_p = N_p^{in} + N_p^{out} := |\mathrm{IN}(p)| + |\mathrm{OUT}(p)|$ the total amount of attached buffers separated into in- and output buffers.*

A function $U_p : S \times \{0,\dots,C\} \times \mathbb{N}^{N_p} \to \mathbb{N}^{N_p}$ mapping states, event count along with input and output buffer levels to changes *in input and output buffer levels is called update function if it fulfills the following condition.*

- *Data is preserved within individual ECAs. It must therefore hold for all vectors of associated buffer levels $b = \begin{bmatrix}(b_i)_{i\in\mathrm{IN}(p)} & (b'_i)_{i\in\mathrm{OUT}(p)}\end{bmatrix} \in \mathbb{N}^N$, event counts $c \in \{0,\dots,C\}$ and states $s \in S$*

$$\sum_{i\in\mathrm{IN}(p)} U_p^{(i)}(s,c,b) = \sum_{j\in\mathrm{OUT}(p)} U_p^{(j)}(s,c,b) \leq c$$

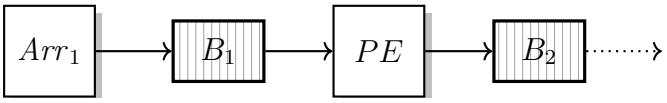


Figure 3.3: *In this sample ECA network, the arrival ECA Arr_1 deposits into buffer B_1. The service ECA PE obtains data from there and delivers them to buffer B_2 where they are further processed.*

Figuratively speaking, an ECA does not store or consume data. It only moves it from buffer to buffer. Typically, the number of moved items is equal to the event count c. It can be smaller, however, if the buffers have less items, for instance.

Definition 3.2 (ECA buffer)**.** *A buffer B is represented by its fill level b. It evolves according to incoming messages $U_{in}^{(B)}$ and outgoing messages $U_{out}^{(B)}$. Here $in = \{p : \mathrm{OUT}(p) \ni B\}$ refers to the unique ECA who deposits into B. $out = \{p : \mathrm{IN}(p) \ni B\}$ analogously refers to the unique ECA that reads from B.*

$$b^+ = \max\left[0, \min\left(b + U_{in}^{(B)} - U_{out}^{(B)}, B_{\max}\right)\right] \tag{3.3}$$

The $(\cdot)^+$ is a temporal next operator. More precisely, if $b = b[k]$ refers to the content of the buffer at the end of slot k, $b^+ = b[k+1]$ refers to the content one time step later[1].

Note that, while the ECAs cannot lose data, the buffer mechanics in (3.3) let the buffers throw away data if they run full. This mirrors real-life buffer behavior. If one wants to check whether a certain buffer capacity $B_{\max}$ is sufficient to keep all data, however, one has to model buffers that are larger than $B_{\max}$ and check whether this level is exceeded at any time.

Fig. 3.3 shows an exemplary ECA network that operates in this fashion. In this system, $\{A_p\} = \{Arr_1, PE\}$ and $\mathcal{B} = \{B_1, B_2\}$. *IN*, *OUT* are clear from the graph; $B_{\max}, C$ can be specified as circumstances demand.

With the main parameters in place, we now look at the update functions $\{U_p\}$. For the arrival automaton, the update function is simply

$$U_{Arr_1}(s, c, b) = c. \tag{3.4}$$

Similarly, the update function of PE is given by

$$U_{PE}(s, c, b) = \begin{bmatrix} c & c \end{bmatrix}. \tag{3.5}$$

In other words, both ECAs consume and deliver their event count directly.

A more complicated example that demonstrates Time Division Multiple Access (TDMA) arbitration with ECAs is in Fig. 3.4. It begins on the left with a set of arrival ECAs. These automata deposit all their events into the single attached buffer using the simple update function from (3.4).

[1] This is a convenient notation because it emphasizes that there is no interaction between more distant time steps and it translates well into the implementation language for the Binary Decision Diagram (BDD)-based model checking tool SAL [54].

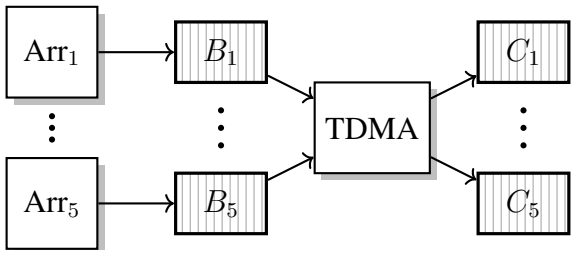


Figure 3.4: *This ECA network demonstrates a single service ECA handling multiple streams. This raises arbitration questions that must be handled by its update function.*

From the first buffers, the data is further processed by a single time-triggered ECA *TDMA*. This service automaton implements a time-triggered scheduling policy, i.e., it associates each state with a certain input buffer. Cycling through the states s_j in round-robin fashion, it processes exactly the buffer the current state is assigned to. If there is nothing to be processed in this buffer, the processing opportunity keeps unused. This protocol is realized by the following update function.

$$U^{(i)}_{TDMA}(s_j, c, b) = \begin{cases} \begin{bmatrix} c & c \\ 0 & 0 \end{bmatrix} & \text{, if } i = j \\ & \\ \begin{bmatrix} c & c \\ 0 & 0 \end{bmatrix} & \text{, if } i \neq j \end{cases} \tag{3.6}$$

3.3 Methods for evaluating ECAs

We have now explored the theory behind ECAs. Next, we add augmentation methods that permit convenient evaluation of the common questions that arise in the design of communication networks. This passage first shows how the maximum delay for individual message streams can be calculated and then treats the evaluation of individual deadline misses that are needed to collaborate with the LTL properties from Section 2.4.

Delay calculation Once an ECA network is set up, the maximum delay for a specific message stream can be calculated in the following way. Initialize an integer array $D_j = 0$. D_j tracks how many messages have been in the system for j units of time or more. With the buffer mechanics and the next-operator $(\cdot)^+$ explained in Definition 3.2 in mind, update D_j to D_j^+ as follows. Here, $\mathrm{inc}(b_1)$ refers to the increment of the first buffer, i.e., the new arrivals; $\mathrm{dec}(b_{\text{last}})$ to the decrement of the last buffer, i.e., the messages leaving the system. Both are to be regarded with respect to one specific message stream.

$$D_j^+ = \begin{cases} \mathrm{inc}(b_1) + D_1 - \mathrm{dec}(b_{\text{last}}) & \text{, if } j = 1 \\ \max\left(0, D_{j-1} - \mathrm{dec}(b_{\text{last}})\right) & \text{, otherwise} \end{cases}$$

The maximum delay that a message of this stream can experience is then bounded by $\max\{j : D_j > 0 \text{ at any time}\}$. In a model checking environment, this value can be calculated in the

following way. Initialize a variable $d = 0$ and update d according to

$$d^+ = \begin{cases} d & \text{, if } D_{d+1} = 0 \\ d+1 & \text{, otherwise.} \end{cases}$$

Note that this model does not delete buffer overwrites from the delay count. An overwritten message will therefore cause an unbounded worst case delay for the concerned message stream. This is the correct interpretation in most cases, but it is not suitable for the evaluation of deadline firmness that we discuss next.

Verification of deadline firmness The (f,H)-firm deadline interface from Section 2.4 cannot be evaluated by merely asking for the maximum delay. This worst case delay may in many cases be unbounded and therefore not reveal any information. In a typical distributed CPS situation – the assumptions will be detailed shortly – the validity of a (f,H)-firm deadline can be checked by introducing an evaluation helper automaton for counting failures and an array to store them.

The evaluation automaton is depicted in Fig. 3.5. Its inputs are clk_{arr} and buf_{total}. These refer to the clock in the corresponding arrival automaton from the stream under scrutiny and the total amount of messages that are inside the system for this stream. In the ideal case, a message arrives in cycle $\mathrm{clk}_{arr} = 0$. The automaton stays in "neutral" until the deadline or threshold delay τ_{th} is reached. It then counts the messages that the stream currently has inside the system. If the deadline is shorter than the period, i.e., if no new message has been inserted into the stream and the last message has arrived, there should be no messages inside the system. In this case, the automaton moves to "success". Otherwise, it shifts to "fail".

Evaluating communication with the automaton from Fig. 3.5 relies on the following assumptions.

1. Messages in the stream under discussion arrive periodically, possibly with small jitter, but without burst. This is necessary for the arrival automaton's clock to serve as clock for the evaluation automaton.

2. Messages are only overwritten by other messages from their own stream and they cannot bypass each other.

3. Messages have a deadline shorter than their arrival period.

These assumptions hold in virtually all control systems. It is very rare that the sampling period is chosen to be smaller than the specified deadline. If a message would bypass another, it should rather overwrite its predecessor since the state information that is transmitted is not sequential. In other words, if the last state is knows, knowledge about the older states is no longer needed to actuate the system. Under these assumptions it is sufficient to test that the stream has no more messages inside the system when the clock of the arrival automaton reaches the deadline. When no messages are inside the system, the last message that was sent must have successfully arrived since – being last – it could not have been overwritten.

Reading inputs from multiple distributed sensors may necessitate merging streams. This does not constitute a major challenge for the approach from this section. Having a message inside the system when it is evaluated at the deadline still represents a failure.

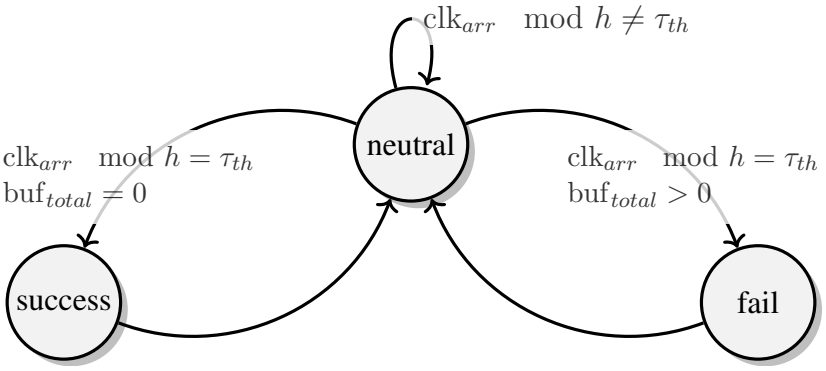


Figure 3.5: *Evaluation Automaton with inputs* clk_{arr}*, the clock of the corresponding arrival automaton,* $\mathrm{buf}_{total} = \sum_{i:i\in\mathrm{Stream}} b_i$*, the total amount of messages the stream of interest has inside the system.*

With the evaluation automaton in place, checking for (f,H)-firmness (Definition 2.7) is possible by introducing an array of integers. Consider first the simpler case of a $(m+1,1)$-firm deadline that requires only one integer. In this situation, we wonder whether it is guaranteed that no more than m consecutive messages fail to meet their deadline. To track the maximum number of uninterrupted deadline misses, we initialize a variable $\mathrm{failcnt} = 0$. We then update it with $(\cdot)^+$ the temporal next operator depending on s, the state of the evaluation automaton.

$$\mathrm{failcnt}^+ = \begin{cases} \mathrm{failcnt} + 1 & \text{, if } s = \mathrm{fail} \\ 0 & \text{, if } s = \mathrm{success} \\ \mathrm{failcnt} & \text{, if } s = \mathrm{neutral} \end{cases} \tag{3.7}$$

To guarantee that there are at most m message drops after every successful transmission, it then suffices to model check for the LTL formulation

$$\mathbf{G}(\mathrm{failcnt} < m).$$

For general (f,H)-firm deadlines as discussed in [91] and Section 4.1, a fail count array F of size H is updated in a similar way. Its elements F_j represent the number of failed transmissions over H time steps. They differ in their offset; F_0 counts messages in the interval $[0, H-1]\%H$, F_1 is associated with $[1, 0]\%H$. $\%H$ is modulo H and indicates that time steps are tracked in round-robin fashion. At the end, it must be verified that none of the array's elements exceeds f at any time. Note that it is significantly easier to evaluate the number of messages in the system than to track the delay of individual messages as in [91].

Timestamping messages The techniques presented for deadline evaluation so far all cater to one specific question. This keeps their implementation nimble with respect to the overall state space size implied by an ECA network. In certain cases, we may want even more precise information about the behavior of the network. Tracking individual patterns of message delay tends to be difficult with the discussed approaches, for instance.

One has to keep in mind that the ECA framework was designed to answer questions like "Is there a buffer overflow?" or "What is the maximum end-to-end delay of a stream?". The

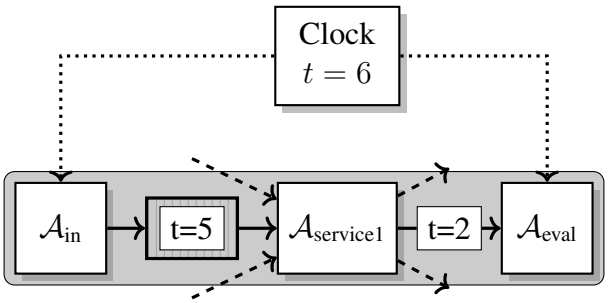


Figure 3.6: *A stream of time-stamped messages in a network of ECAs*

buffers therefore do not hold individual messages; only the number of messages that are present in a specific buffer is stored. Once a message is in the network, we do not know when it arrived anymore, in general. The automaton from Fig. 3.5 alleviates this only under certain assumptions and not for general delay-related questions. The more general approach of timestamping hence proves valuable in practice.

By adding timestamps to messages, it becomes possible to answer very general questions about delay. If there are many such questions, this may be more efficient than adding many custom-designed evaluation automatons. Since it does incur a significant overhead, however, it is advisable to track only individual streams in this manner. In addition to system analysis, timestamps could also be used for scheduling inside the system. Removing messages from the bus that can no longer meet their deadline could be an efficient scheme, for instance.

The overall situation of a supervised stream is shown in Fig. 3.6. It is started by an arrival ECA and ends with an evaluation automaton. In parallel to this, a global clock is introduced that runs in round-robin fashion to keep the state space finite. This clock can in certain cases be the clock of an arrival automaton. Messages that go into the system are stamped with the value of the clock at that instant. While messages make their way from buffer to buffer through the system, their progress is tracked and finally their stamp is compared to the global clock once they are through the network. Here, we see an old message with $t = 2$ and a more recent message with $t = 5$ that just left the arrival ECA. In between, they are processed by a shared service ECA or possibly a sequence of service ECAs; the dashed connections indicate competing streams.

There are many ways to implement timestamping in ECAs. The approach in this work is inspired by the semantics of SAL. In the following, we exclude merging streams from the part of the ECA network that we want to validate and discuss only buffers of size one. This corresponds to the requirements of control applications and significantly simplifies the necessary discussion. For typical streaming applications, like audio/video streaming, larger buffers and merging streams are common, however. In order to consider merging streams, it may be sufficient to tag the outgoing merged message with the time of the older ingoing message after the merging service ECA. Larger buffers may necessitate storing arrays of index variables. This does not necessarily deteriorate performance because it must not lead to additional states in the back end. The biggest issue there is branching and such arrays would not introduce new optionality into the system. Both these topics should be examined more closely, however.

In the SAL framework, and possibly in other model checking tools as well, one solution

is tracking the timestamps in the evaluation automaton directly. The buffers then pass around indexes that refer to the timestamps managed by the evaluation automata. In an object-oriented environment, one would rather extend the messages with this data and pass them around as larger data structures. This, however, can be difficult in the available modeling tools.

Concretely, the messages that enter the system are numbered in round-robin fashion. The buffers store an index that marks the position of the messages it currently holds with respect to all the messages in a stream array. The buffer mechanics from Definition 3.2 are extended as follows.

The first buffer B_1 behind the arrival automaton works as follows. It is initialized with index $i[0] = 0$ and empty, i.e., $b[0] = 0$. It is then updated via

$$\begin{aligned} b^+ =& \max\left[0, \min\left(b + U_{in}^{(B)} - U_{out}^{(B)}, B_{\max}\right)\right] \\ i^+ =& (i + U_{in}^{(B)}) \bmod i_{\max}. \end{aligned} \tag{3.8}$$

As before, the buffer elements are increased by new arrivals $U_{in}^{(B)}$. These are messages coming into B from the automaton where B serves as output buffer. $U_{out}^{(B)}$ similarly refers to the messages leaving toward the ECA for which B is an input buffer. The addition is the evolution of the index. The buffer simply increments this variable by the number of messages that are deposited. In other words, i corresponds to the index of the newest message inside the buffer and together with b, one can deduce all indexes.

Consider now B_2, a buffer that comes later in the stream. There, it has to be taken into account that messages may be overwritten along the way and never arrive in B_2. This is achieved by the following mechanics.

$$\begin{aligned} b^+ =& \max\left[0, \min\left(b + U_{in}^{(B)} - U_{out}^{(B)}, B_{\max}\right)\right] \\ i^+ =& \begin{cases} i_{in} & \text{, if } U_{in}^{(B)} > 0 \\ i & \text{, otherwise} \end{cases} \end{aligned} \tag{3.9}$$

In these buffers, the index remains constant unless a new message arrives. If that is the case, the index is overwritten with the index of the newcomer. This corresponds to overwriting old information in the buffer that was rendered useless by the last arrival.

Fig. 3.7 contains an evaluation automaton for evaluation of deadline firmness patterns. Instead of evaluating the number of messages in all buffers of the stream as in Fig. 3.5, the delay of the last arrival is now evaluated directly. The states of this automaton can once more be tracked by LTL formulas (see Section 2.6).

Runtime comparison of timestamping and direct deadline firmness verification Timestamping provides more freedom in terms of evaluation. At the same time, it also creates more states and is thus computationally more expensive. To quantify this difference, a runtime evaluation was performed using the platform from [91], reproduced in Section 4.1. This comparison considers the time required for a deadlock-check where the full state space must be constructed and traversed. Individual properties can terminate early and are thus harder to compare.

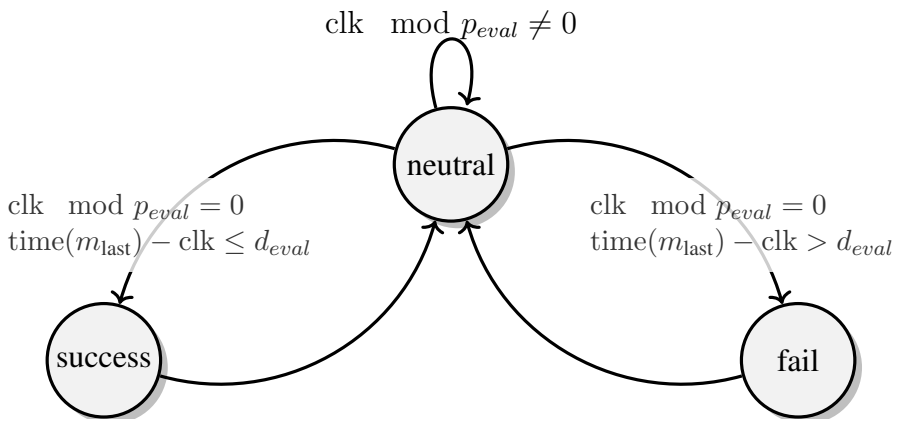


Figure 3.7: *Given that arrival times are captured in an array* TIME *by the ECA network, "success" and "fail" states can be emitted based on the delay of the last message to leave the system.*

The deadlock-check required 45s with the timestamping technique. The direct method from (3.7) for the special case of a (1, 1+H)-firm deadline requires less than 2s. All these measurements were performed on a workstation with INTEL I7-3370 CPU @ 3.4GHz and 16GB RAM.

This demonstrates that specific evaluation automata may be well worth the effort over timestamping. The results may be less convincing, however, once the property asks for more general (f,H)-firm deadlines. These results were also reported in [90].

3.4 ECAs in the model checking tool SAL

We now have the theory in place for ECA modeling. Going from there, the next paragraphs are supposed to offer a glimpse of how these models are subsequently implemented. In this work, the tool selected for this purpose is SAL [54], but there are several similarly suitable alternatives. NUSMV is similar to SAL, for instance, and also performs model checking on Binary Decision Diagrams (BDDs). As such, it could also serve as back end for ECAs. The original work from [41] implements its examples in CPN TOOLS [84]. Fore more information on Colored Petri Nets (CPNs), please refer to [83].

The advantage of SAL is that its syntax can conveniently construct parallel elements, like the buffers in Fig. 3.4, in an array-like container. This allows resizing case studies conveniently and appears to be unique among current open-source model checking tools.

SAL models are written in functional programming fashion. Its style is similar to the SCHEME language that the SAL scripts themselves are implemented in. The basic idea we pursue for ECA implementation is the following. We implement the buffers and each ECA as a *module*. We then execute these modules in parallel using the "||" operator.

Recall the arrival ECA with only two states from Fig. 3.2. This automaton has been implemented in SAL as shown in the following listing. The apostrophe in, e.g., k' is the *next operator* in SAL. It works just like the $(\cdot)^+$ operator described in Definition 3.2.

Listing 3.1 describes the two states of the ECA individually. The implementation thus re-

Listing 3.1: SAL *code implementing the arrival ECA from Fig. 3.2.*

```
eca_example : MODULE =
States : TYPE = { A, B };
BEGIN
LOCAL     s : States, x : [0..2]
OUTPUT    k : [0..2]   %event count
INITIALIZATION
  s = A; x = 0; k = 0;
TRANSITION
  [ s = A -->   %state A
    %choose k' from set containing d in {0,1} s.t. x+d <= 1
    k' IN { d : [0..1] | x + d <= 1 };
    s' = B;   % move to B
    x' = x + k';  % update x
  [] else -->   %state B
    %choose k' from set containing d in {1,2} s.t. x+d = 2
    k' IN { d : [1..2] | x + d = 2 };
    s' = A;   % move back to A
    x' = 0;     % reset x
  ]
END;
```

mains close to the graphic representation of the ECA.

Listing 3.2 contains an example for an LTL formulation, referred to as theorem in the SAL language. Specified systems are evaluated with respect to such theorems. SAL replies whether they are guaranteed to hold or returns a counter-example for the specification.

Listing 3.2: *The LTL formulation from* (2.29) *has been implemented in* SAL *for a (2,5)-firm deadline.*

```
wmodabs(a, base: NATURAL) : NATURAL = IF a>=0 THEN a ELSE a+base ENDIF;

max2in5 : THEOREM system
  |- (FORALL (vcyc:ecyc_t) : G ( (ecyc=vcyc) =>
    W( NOT(s=tx_f), wmodabs(ecyc-vcyc,ecycles)=cyc_5 OR (s=tx_f AND X
      [W( NOT(s=tx_f), wmodabs(ecyc-vcyc,ecycles)=cyc_5 OR (s=tx_f AND X
        [W(NOT(s=tx_f),wmodabs(ecyc-vcyc,ecycles)=cyc_5)]
      ))]
    ))
  ));
```

3.5 Issues with naive Fixed-Priority Non-preemptive Scheduling (FPNS) models

We have seen basic buffer mechanics for ECAs in Fig. 3.3 and time-triggered arbitration in Fig. 3.4. Event-triggered arbitration, like FPNS, is more intricate, however.

Following [92], we will first describe a basic FPNS model implemented in the ECA framework. Its mechanics are very close to those of a simulation implementation in, e.g., SYSTEMC. Yet, the time granularity of an ECA model must be chosen much coarser than that of a simulation to avoid state space explosion. Therefore, this intuitive model cannot capture all the intricacies of a real-world CAN bus. We will first explore these effects and see how they can lead to overly optimistic bounds in detail. Since the main purpose of ECAs is system verification, optimistic bounds are not helpful. Relying on such bounds can and will lead to deadline misses and, as a consequence, to potentially severe consequences in the real world.

After analyzing the shortcomings of the straightforward model, we discuss conservative, i.e., safe models in Section 3.6. These models extend the ECA syntax to differentiate between messages that newly arrive in a buffer and those that had been there for a longer time. Finally, we motivate their use in a case study in Section 3.7.

Straightforward FPNS model The most basic implementation of FPNS, that is closest to how a simulation tool would implement an abstract CAN bus, would employ a constant service automaton (one state A with constant rate $\rho(A) = [c, c]$) and combine it with a priority-based update function where $\pi(b)$ is the priority of a certain message buffer. Consider an update function for input buffers $\{i\}$ and output buffers $\{i'\}$ where $U^{(i)}(s_i, c, b) = U^{(i')}(s_i, c, b) = g_i$ and

$$g_i = \max\left(0, \min\left(b_i, c - \sum_{\substack{j \in \mathrm{IN}(p) \\ \pi(b_j) > \pi(b_i)}} b_j\right)\right). \tag{3.10}$$

This lets the PE handle all the data in the higher priority buffers before moving on to lower priority items. Note that for the highest priority buffer it holds $U = \min(b_i, c)$, meaning it always gets the full service unless there is not enough data in the buffer.

Timing Issues of the Straightforward FPNS Implementation The straightforward FPNS model described by update function (3.10) assumes that the messages on the FPNS bus are aligned to the ticks of the ECA as illustrated in Fig. 3.8. The arbitration in the model takes place at the beginning of every tick and it is impossible to resolve timing issues within a slot. More precisely, if two messages arrive in the same slot, it is impossible to say which one was first. Aligning the ticks of the ECA realm with the real-world timing of, e.g., the CAN bus is an overly simplistic assumption, however, and cannot hold in practice. When the bus is idle, it does not wait for the next time slot to begin before transmitting once a message is in the buffer ready to be sent. Such a behavior might make verification easier, but it is certainly undesirable when optimizing for throughput.

On the other hand, the fundamental time unit of the CAN bus is dramatically smaller than an ECA model can capture without rendering its state space unmanageable. It is therefore common that messages arrive and processing occurs without being aligned to the ECA time slots.

CAN priority inversion The CAN bus can have situations where higher priority messages are blocked by lower priority messages. Such cases occur when the lower priority message arrives first and starts sending before the higher priority message arrives. At first glance, the important

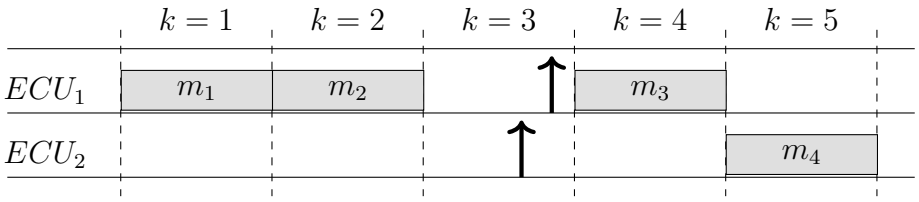


Figure 3.8: *Perfectly aligned timing as assumed by the straightforward FPNS model ignores blocking by lower-priority message. Table 3.1 shows the corresponding ECA trace. The arrows denote the actual message arrival in the send-buffer.*

information is only delayed during this single transmission. It could get held up indefinitely, however, if messages of even higher priority arrive in the meantime.

Priority inversion is not captured by the straightforward model and leads to over-optimistic bounds when analyzing it in a model checker. To see this issue in more detail, let us assume for simplicity that there are only two message streams and that they are indexed in order of descending priority (1 being the highest priority). Further let the underlying constant service automaton process one message per time slot, i.e., $c = 1$.

As shown in Fig. 3.9, after two messages, m_1 and m_2 from the buffer for ECU_1 are sent, all the buffers are empty and the bus idles at the beginning of $k = 3$. Then, a message arrives for ECU_2 (indicated by the black arrow) and the bus starts processing it immediately since ECU_2 is the highest priority accessing it at that moment. When ECU_1 wants to transmit data shortly after (black arrow), the bus is already utilized and ECU_1 is blocked (transparent) until ECU_2 finishes its uninterruptible message m_3. Only then is m_4 transmitted.

Let us compare this to how the straightforward FPNS model would process this arrival pattern. Fig. 3.8 contains the same arrivals, but the messages are processed according to the straightforward model. The evolution of the two buffers b_1 and b_2 under this model is further detailed in Table 3.1. It starts out just like the real pattern and handles the two items in the buffer of ECU_1. At the end of $k = 2$, the buffers are empty and nothing is processed in $k = 3$, but there are two arrivals.

Table 3.1: *Buffer fill levels at the end of individual steps k showing the behavior of the basic FPNS model for the arrival pattern in Fig. 3.8 and Fig. 3.9.*

k	0	1	2	3	4	5
b_1	2	1	0	1	**0**	0
b_2	0	0	0	1	**1**	0

The model starts to differ from the observed pattern in step $k = 4$. Here, it would just process the higher priority message because it performs arbitration at the beginning of the slot only. This has been marked in bold in Table 3.1. The bus, however, would have started processing the message of ECU_2 already before the beginning of slot $k = 3$ as described above. This would correspond to $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$ at the end of $k = 4$. Calculating the maximum delay for ECU_1 from this model, e.g., by using the method from Section 3.3, and using it as a bound

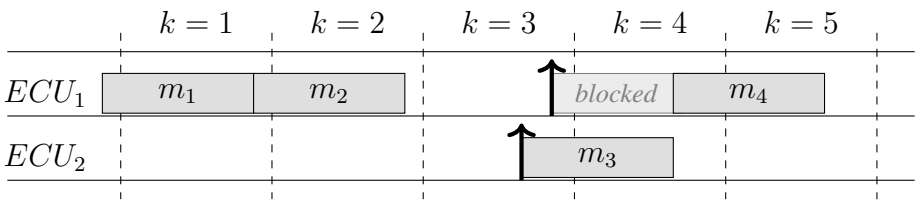


Figure 3.9: *Timing diagram illustrating how a higher priority-message can be blocked by a lower-priority one in FPNS scheduling. Fig. 3.8 and Table 3.1 show the corresponding sequence in the basic ECA implementation that fails to capture this effect.*

would therefore be overly optimistic. As we will see in the case study in Section 3.7, such a bound will be violated by some of the messages.

3.6 Conservative FPNS models

Blocking by lower-priority messages as described in Section 3.5 does occur and needs to be taken care of at the ECA modeling level. We will start by showing how a model could look like for an FPNS element that processes one message per tick (or less, e.g., one message every 3 steps) and then extend it to a multi message model. Processing more messages per slot reduces the timing accuracy of the ECA, but also leads to less states and thus potentially to faster computation.

In this section, we will introduce a model that takes the possible blocking by lower priority messages into account when a transmission starts before the slot it ends in. Towards this, we need to consider the various paths the intra-slot behavior could take due to the underlying slotted-time mechanics. These paths then need to be included into the update function that governs the arbitration between the individual message streams. The example from Fig. 3.9 and Table 3.1 showed that we cannot just take the highest priority from the buffer and process it. Instead, we have to consider that any newly arrived message with even higher priority could take its place. Fig. 3.10 illustrates this in greater detail.

With one message in the buffer for ECU_3 at the end of $k = 1$ and one message arriving for both ECU_1 and ECU_2 during $k = 2$, the discrete-time semantics allow for several scenarios. Each scenario consists of two arrivals (arrows) and one message (labeled a, b, c) and is further described in the following paragraphs.

(a) The message from ECU_1 arrives before the transmission of m_1 ends and it therefore gets to send next. It is irrelevant when the message of ECU_2 arrives in this case.

(b) The message from ECU_2 arrives before m_1 is entirely transmitted and ECU_1 is ready to send only afterwards. ECU_2 therefore sends next.

(c) Both ECU_1 and ECU_2 get ready to send after the transmission of m_1 finishes. ECU_3 therefore transmits next.

Since the slotted time does not hold enough information to resolve which message arrives first within the time slot, all the possibilities need to be accounted for.

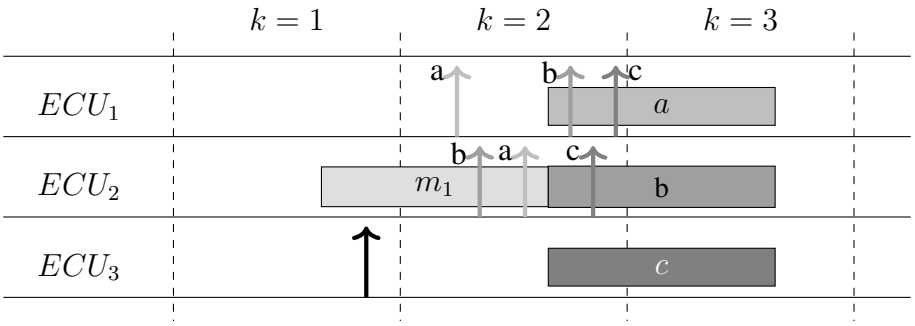


Figure 3.10: *Three scenarios (a,b,c) with identical arrival signature from the ECA's point of view lead to different outcomes when the CAN's arbitration is not aligned to the ECA's ticks in the single message case.*

Enhanced buffer mechanics To calculate all these possibilities, we need to track which messages were already present at the beginning of the slot and which have just arrived during the last step. We therefore extend the basic buffer semantics from Definition 3.2 where only the total buffer content was tracked. In addition to b, the total elements in the buffer at the end of the slot, we now track n, the elements that newly arrived during this slot. This results in

$$\begin{aligned} n^+ =& U_{in}^{(B)} \\ b^+ =& \max\left[0, \min\left(b + U_{in}^{(B)} - U_{out}^{(B)}, B_{\max}\right)\right]. \end{aligned} \tag{3.11}$$

The elements that remained in the buffer after all the messages from the last step were sent are then given by

$$r^+ = b^+ - n^+. \tag{3.12}$$

Selecting one message per ECA cycle for processing With this additional and more fine-grained information we can now select the message i_0 that the bus starts sending at the end of the last step and finishes in the current step. As shown in Fig. 3.10, this can be either the message with the highest priority that was already in the buffer and whose transmission therefore would have started immediately after the bus was idle (scenario (c)) or any of the newly arrived messages that have higher priority (scenarios (a) and (b)). This is summarized in the following options.

$$i_0 \in \left\{ i : \underbrace{b_i > 0}_{\text{has message to send}} \quad \text{AND} \quad \underbrace{r_j = 0 \quad \forall j > i}_{\text{no higher priority is waiting}} \right\} \tag{3.13}$$

Such a diverging path can be easily included in the model checking tool SAL.

Next, the corresponding update function is adjusted. With i_0 from (3.13) in mind, let

$$g_i := \begin{cases} 1 & \text{, if } i = i_0 \\ 0 & \text{, otherwise.} \end{cases} \tag{3.14}$$

This constitutes the new update function $U^{(i)}(s, c, b) = U^{(i')}(s, c, b) = g_i$, where i and i' refer to the in- and output buffer of stream i, respectively.

The single message model extends well into a more fine-grained time discretization, where one message is transmitted every couple of time steps. Here, one would just change the underlying service automaton and leave the arbitration as described when there is service to distribute. Additional delay could be modeled by introducing new buffer stages that the message has to travel through.

Processing multiple messages in a single ECA cycle As discussed, extension of the single message FPNS model from (3.13) towards finer timing is straightforward. Establishing a similar model for even less fine-grained timing where multiple messages are sent during a single time slot still poses a challenge, however. One possibility for modeling the behavior in this case is to iterate the selection process for i_0.

Start by assuming that i_0 was chosen in the previous time step using the rule from Eq. (3.13). The next message to be transmitted faces a situation that is similar to the single message case shown in Fig. 3.10. The difference in the selection of i_1, the ECU to send after i_0, lies in the available messages. Firstly, the messages that were in the buffer at the end of slot $k = 1$ are available if they were not sent already, i.e., minus i_0. Additionally, all the messages with higher priority that might have arrived by then could be ready to send. Since, again, it is impossible to distinguish arrivals within a single time step, this refers to all the messages arriving in the respective slot. Therefore i_1 has to be selected as

$$i_1^+ \in \left\{ i : \underbrace{b_i + n_i^+ - \delta_{ii_0} > 0}_{\text{potentially has message}} \text{ AND } \underbrace{r_j - \delta_{ji_0} = 0 \quad \forall j > i}_{\text{no higher priority}} \right\}$$

where δ_{ij} refers to the Kronecker delta

$$\delta_{ij} = \begin{cases} 0 & \text{, if } i \neq j \\ 1 & \text{, if } i = j \end{cases}.$$

After i_1 has been determined, further i_l can then be calculated iteratively:

$$\begin{aligned} i_l^+ \in \Big\{ i : \quad & b_i + n_i^+ - \sum_{m=0}^{l-1} \delta_{ii_m} > 0 \\ & \text{AND } r_j - \sum_{m=0}^{l-1} \delta_{ii_m} = 0 \quad \forall j > i \Big\} \end{aligned} \tag{3.15}$$

3.7 Evaluation of FPNS models using simulation

For a demonstration of the benefits from the models developed in Section 3.6, consider the following case study performed using SAL and SYSTEMC. The abstract ECA networks from both the straightforward model and the proposed extended models are transformed to their transition system representation. Next, they are implemented in the modeling language of SAL. Finally, the BDD-based model-checker in SAL verifies the respective system behavior.

We calculate bounds on the delay for every stream via the method outlined in Section 3.3. These bounds are referred to as *StraightForw*, *SingleMsg* and *MultiMsg*, respectively. Additionally, a SystemC model for the CAN bus is constructed. Observing a simulation there for $T = 10\,\mathrm{Ms} \approx 2.7\,\mathrm{h}$ yields the average delay *Avg* and the maximum delay *Max*.

Finally, we perform a manual analysis of the worst case scenario where all messages start at the same time and are initially blocked by the longest lower priority message. This is denoted by *Manual*. It is both tedious and error-prone and, unlike ECAs and SYSTEMC, this analysis cannot be applied to state-based or even just larger systems.

The experiment assumes 4 message streams numbered #1 to #4 from highest to lowest priority with periods

$$P = \begin{bmatrix} 1\,\mathrm{ms} & 2\,\mathrm{ms} & 4\,\mathrm{ms} & 5\,\mathrm{ms} \end{bmatrix}$$

and a common jitter of $j = 0.5\,\mathrm{ms}$. Here, jitter refers to enlarging the scheduled arrival $k \times P_i$ to an interval $\left[k \times P_i, k \times P_i + j\right]$.

The message streams share a CAN bus with transmission time of $0.5\,\mathrm{ms}$ per message. No state dependency was modeled in order to have readily available reference solutions for comparison.

Runtime Evaluation The model checking runs for the following case study took $16\,\mathrm{s}$ for the single message model and $85\,\mathrm{s}$ for the multi message model on an Intel i7 at $3.4\,\mathrm{GHz}$ with $16\,\mathrm{GB}$ RAM. Note that while the multi message model took longer to verify in this case, it might be the preferred or only choice in case another PE dictates a certain slot time.

Single message model For testing the single message model, the ECA slot time is set to $0.5\,\mathrm{ms}$ in accordance with the transmission time. The resulting bounds created by the formal verification of the ECA models via the model checking tool SAL are listed in Table 3.2 as *Straightforw* and *SingleMsg*, respectively.

The same architecture is also modeled in the SYSTEMC simulation framework. We executed the model there and recorded the delays of the individual messages. Fig. 3.11 shows the delay distribution of Stream #3. It illustrates that some messages do in fact incur a larger delay than $2.5\,\mathrm{ms}$, the bound calculated using the straightforward model. By contrast, no message exceeded $4.5\,\mathrm{ms}$, the bound guaranteed by SAL when verifying the newly developed single message model.

Average and maximum delay of the other streams is summarized in Table 3.2. The straightforward model also fails for stream #2. It does hold for streams #1 and #4, however. This is expected behavior because the straightforward model does not miss blocking of the lowest priority messages and the highest priority stream can only be delayed for one slot; no indefinite suspension is possible as for the lower streams.

All the bounds from the single message model shown there prove to be tight within the resolution of the ECA. To clarify, consider stream $\#3$ in Table 3.2 as an example. There, we observe a maximum delay of $3.63\,\mathrm{ms}$ (equivalent to $4\,\mathrm{ms}$ or 8 slots, rounded up for the ECA) and the manual analysis yields a worst-case delay of $4.0\,\mathrm{ms}$ that can occur in practice. Verifying the ECA model with SAL leads to an upper bound of 8 slots. This corresponds to the same $4.0\,\mathrm{ms}$ and would therefore be tight as well. However, we additionally account for the

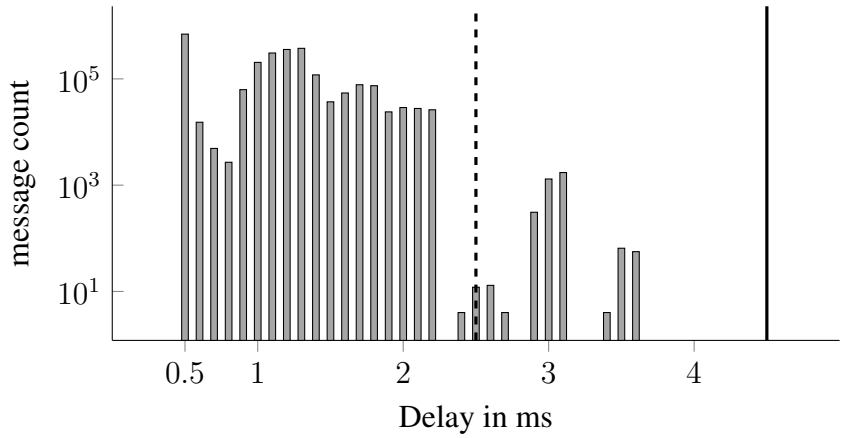


Figure 3.11: *Histogram of delays experienced by message stream #3 as measured in the SystemC simulation. The bound from the straightforward model (dashed, Section 3.5) is exceeded by a significant part of the messages while the bound from the single message model (solid, Section 3.6) holds.*

fact that this worst-case message could have arrived early in slot $k = k_0$ and was transmitted late in slot $k = k_0 + 8$, increasing the upper bound *SingleMsg* by an additional slot to $4.5\,\mathrm{ms}$.

Multi message model We use the same experimental setup for evaluating the multi message model where messages are selected using rule (3.15). Toward this, the ECA slot time is adjusted to $1\,\mathrm{ms}$, such that two messages can be transmitted per slot. Implementing this model in SAL and running its BDD-based model checker yields the bounds *StraightForw* and *MultiMsg* from Table 3.3. These bounds appear to be very conservative with respect to the simulation results and the analysis from Table 3.2. This is due to the higher implied jitter in the multi message case. Since the slot time is $1\,\mathrm{ms}$, the ECA framework must now assume that all the messages suffer from a minimum jitter of $1\,\mathrm{ms}$ as well.

Repeating the simulation and the analysis with this jitter in mind, we obtain the columns *Avg*, *Max* and *Manual* in Table 3.3. Again, we observe that the straightforward model is overly optimistic. In contrast, the proposed model successfully bounds the transmission delay and it is tight within the selected resolution of the ECA framework when compared to the analytical worst case. We could, however, not observe the worst case for all message streams in the SYSTEMC simulation.

Table 3.2: *Single message model FPNS case study results.*

	Delay from SystemC		Analytic Delay Bounds		
Stream #	*Avg*	*Max*	*Manual*	*Straightforw*	*SingleMsg*
1	0.66 ms	0.98 ms	1.0 ms	1.0 ms	1.5 ms
2	0.83 ms	1.83 ms	2.0 ms	1.5 ms	2.5 ms
3	1.10 ms	3.63 ms	4.0 ms	2.5 ms	4.5 ms
4	1.17 ms	3.65 ms	4.0 ms	4.5 ms	4.5 ms

3.8 Related work

ECAs are not the only tool for timing analysis and this work does not newly introduce this automaton framework either. This section lays out the origins of Real-time Calculus (RTC). The difficulties with modeling state dependency there motivate the ECA framework.

Next, the main works in the ECA domain are presented. They form the background for this chapter's contributions, the FPNS model from Section 3.6 and the majority of the CPS-inspired evaluation techniques from Section 3.3. The value of ECAs in the co-verification context is explored at the end of the corresponding chapter, in Section 4.3.

Finally, less closely related automatons and timing analysis techniques are presented. This background knowledge underlines the value of RTC and its extension, the ECA framework.

Network Calculus (NC) The NC scheme is one of the building blocks of RTC. It deals with streaming networks and extends or complements queuing theory there. The applications range from local audio and video transcoding to large networks of an Internet service provider. The goals are finding appropriate buffer sizes, adequate computing or communication elements and discovering bottlenecks in larger graphs with multiple routes.

The framework is purely algebraic and compositional in nature. System inputs are characterized using suitable monotonic functions, the so-called upper and lower arrival curves. The processing elements like the processors, buses and switches are described by service curves, similar monotonic functions. Both arrival and service curves contain information of the system from an interval-based, sliding window perspective. LeBoudec and Thiran in their reference textbook [107] describe this perspective as follows. "For example, if [an arrival curve] $\alpha(t) = rt$, then the constraint means that, on any time window of width τ, the number of bits for the flow is limited by $r\tau$."

The system is then solved by *iteratively* evaluating processing elements via application of the convolution operator on their service curve and the corresponding arrival curves. This compositional approach leads to excellent scalability.

One criticism of the *deterministic* NC where both arrivals and service are bounded is that the bounds may become too loose to be useful in practice. While "the bounds are tight in the sense that there exist worst-case arrival and service patterns for which the bounds become realizable [...], such worst-case realizations are very unlikely to happen [in large scenarios]" according to [48]. This motivates stochastic network calculus. There, arrival and service curves are of probabilistic nature. One consequently obtains delay guarantees that hold for a desired level of

Table 3.3: *Multi message model FPNS case study results.*

	Delay from SystemC		Analytic Delay Bounds		
Stream #	*Avg*	*Max*	*Manual*	*Straightforw*	*MultiMsg*
1	0.65 ms	1.4 ms	1.5 ms	2 ms	2 ms
2	0.78 ms	2.3 ms	2.5 ms	2 ms	4 ms
3	1.02 ms	4.5 ms	5.5 ms	3 ms	7 ms
4	1.14 ms	5.45 ms	7.5 ms	5 ms	9 ms

certainty. For instance, one may obtain that a bound guaranteeing a certain throughput under a given delay constraint holds with a probability of $(1 - \epsilon)$. Jiang's textbook [86] is one main reference for this approach.

There have been several attempts to unify deterministic NC and stochastic NC, e.g., [85]. It currently appears, however, that they cannot be consistently unified and should be seen instead as separately providing great value [49].

Real-time Calculus (RTC) RTC extends NC for use with real-time systems. It is equally compositional and still built upon the interaction of arrival curves with corresponding service curves that contain information of the system from an interval-based, sliding window perspective. Besides the conversion of task models from the real-time domain to the algebraic format of NC, this movement also rendered the approach more accurate and more usable with the novel tools it spawned.

Thiele first announced the desire for a framework that extends NC in [178]. This paper contains a motivational example of a realistic task system for which the request curve is calculated in polynomial time. In [177], the group develops the advantages of RTC further. In their own words, "[t]hese results substantially generalize and sharpen the bounds obtained in [[178], [107]], as both lower and upper bounds are involved."

After the initial success, the authors have contributed more works that explain and evaluate the new framework more comprehensively. [176] integrates RTC into design space exploration. There, one looks for the best combination of, e.g., Central Processing Unit (CPU) models, scheduling policies from a number of options. RTC is used to describe the performance of each possible combination and integrated into a higher-level solver that looks for the best setup. According to the authors, such optimization techniques had almost exclusively relied on simulation before.

In [39], the capabilities of RTC are compared with the theoretical results from the real-time systems area. Chakraborty demonstrates that RTC can replicate many results that had been individually derived before inside a congruent framework. The same author compares RTC to simulation results in [40]. There, a SYSTEMC implementation with components from a comprehensive library and input traces from several research databases serves as simulation representative. RTC achieves a very tight fit in utilization value over each of several dozens of parameter combinations. The delay bounds are less tight, but remain conservative in the sense that the bounds from RTC surround the simulation values.

RTC is nowadays accessible to a wide audience thanks to Wandeler's "Real-Time Calculus Toolbox" [184] for MATLAB. This software dramatically lowers adaptation barriers compared to, e.g., NC where the relevant tools require a development setup with compiler on the user side. Besides this seminal contribution, his thesis [183], arguably among the most comprehensive and consistent explanations of RTC, also presents Real-Time Interfaces. These are motivated by previous interface automata and allow "a holistic one-step approach to design and analysis of systems, sometimes also referred to as correct-by-construction." They are implemented in stateless assume/guarantee fashion and thus scale more favorably than automata.

Event Count Automaton (ECA) framework RTC delivers precise results in many cases and its compositional design leads to unrivaled scalability. However, it cannot accurately capture *state* information. In streaming networks, it is common for a PE to temporarily halt when its output buffer is full. The original RTC approach cannot capture this in a meaningful way. Other challenging examples are scheduling laws that speed up a PE when its input buffer exceeds a certain level like in Section 4.2.

There are several improvements to the original framework that model specific state dependencies. [79] examines correlated streams that are executed in parallel. Such setups introduce blocking-read semantics and may even influence each other via shared memory. That work deals with the interdependence by introducing OR and ORDER semantics that define how processes can be joined. In [23], Bouillard presents a lightweight method for modeling PEs halting due to full output buffer. By adding a feedback loop to the RTC task graph, she achieves a tighter analysis in these circumstances.

In spite of all efforts, modeling general state dependency remains difficult in RTC and most likely in all algebraic frameworks. Consequently, previous efforts have supplemented the RTC framework with a model checking based extension, the ECA [41]. There, both upper and lower curves are represented by an arrival or a service automaton. The models built in this way can be converted to Büchi automata and are then evaluated using standard model checking tools from the digital domain.

Using ECAs, state-based scheduling becomes straightforward as with all automaton-based frameworks. The strength of the ECA approach compared to other automaton-based frameworks is its great compatibility with RTC. Since they all scale far worse than algebraic techniques, they can often handle only small platforms. Using the bridging technique explained in [145], it is possible to model a subpart of the network as ECAs and then compose it with the remainder of the hardware architecture under consideration modeled in plain RTC. This considerably improves scalability. [144] describes another compositional technique where ECAs are replaced by conservatively over-approximating, but syntactically similar automatons. This further speeds up the analysis.

ECAs have shown to be a good model for capturing the timing properties of streaming applications. This thesis summarizes, for the first time, the extension of their use case to underlying communication platforms of CPSs. The timestamping approach for evaluating arbitrary delay patterns from Section 3.3 has been presented in [91]; the custom approach for (1,H+1)-firm deadlines from the same section stems from [90].

In the CPS context, the CAN bus plays an important role. A model for FPNS, the corresponding arbitration strategy, has been proposed for RTC in [47]. However, it still carries the usual issues with state dependencies that RTC faces. This motivates [92], examining FPNS modeling and its issues for discrete-time automatons with a focus on the ECA framework. This work is reproduced from Section 3.5 onwards.

Others The need for state-based and non-preemptive scheduling has motivated automaton-based methods. [106], for instance, introduces UPPAAL, arguably still the leading tool for verification of Timed Automata (TAs). Refer to [20] for a more recent overview of the TA flavors implemented there. The TA approach defines timed languages that associate a time $\tau \in \mathbb{R}$ with each transition an automaton takes. This leads to a very rich interface but also great

difficulty in evaluating it. Consider [4] for the theoretical background of the original version.

Approaches to FPNS analysis using TAs are presented in [99] and [65]. Because these approaches employ a continuous time model, they track the evolution of the system in a much more fine-grained fashion than necessary in most cases. In addition, they are difficult to interface with algebraic methods, such that scalability often becomes an issue. For instance, the technique from [65] is not applicable when arrival times are given in intervals.

Usually, there is a trade-off between time-granularity and scalability of the modeling. Timed Automata (TAs), for example, intend to track the system's behavior with high precision and often result in models that cannot be solved in practice. Event Count Automata (ECAs), on the other hand, have been developed for streaming applications and their inherent uncertainty. As a result, they naturally model data arrival patterns that need to be scheduled and integrate well with analytic approaches. In other words, the ECA with its unit time intervals strikes a balance between the high precision tracking of the TA and the scalability of RTC. Its fully discrete nature allows timing guarantees by delegating to powerful, established model checking tools originating from the digital design area.

[104] proposes another automaton extension for RTC. It defines "input generators" to obtain networks of TAs from RTC-conforming input curves. It further specifies how to derive conform output curves from the TAs. UPPAAL is used for implementing the whole approach. This idea leads to a framework with high flexibility whose scalability appears to be on par with the ECA method. The difficulty lies in the generation of RTC curves in the "output generators". There, complicated patterns may lead to gross overestimation. Nevertheless, this approach deserves further consideration and can generate the drop patterns that this thesis generates with ECAs.

Common scheduling analysis based on, e.g., demand bound or response time analysis, is still around as well but continues to be challenging. These approaches become pessimistic for the non-preemptive case and are often not compositional in nature. This has given rise to NC in the first place. Furthermore, Davis has recently discovered that commonly accepted analysis techniques for CAN timing from 1994 are flawed [53]. This demonstrates how difficult avoiding human error in custom-made methods with narrow focus really is.

4 Applying ECAs for CPS Co-Design

This chapter presents two applications that combine ECA modeling from Chapter 3 with the control system analysis from Chapter 2 for CPS co-design.

The first application (Section 4.1) has originally been presented in [91]. Given a control performance requirement, it finds patterns of deadline misses that are tolerable. By analyzing these patterns on an ECA model via model checking, the performance of a CPS as a whole can then be verified.

The second application (Section 4.2) has been published in [90]. Here, a fault-tolerant control strategy is designed to match the delay patterns a hardware platform may exhibit. In this way, the CPS as a whole fulfills exponential stability by construction.

Both applications utilize the fact that the average delay in complex platforms is significantly smaller than the worst case. By specifying patterns and not just a single number, a tighter integration is achieved. In this way, the guaranteed performance can be improved because additional options become available in the design space.

4.1 Verification of real-life performance instead of deadlines in distributed CPSs

After discussing checkable CPS requirements in Section 2.6 and the ECA framework for modeling communication platforms underneath CPSs, we can now bring the two together. In the co-verification approach of this section, the goal is to expand the typical deadline specification interface. The overall technique yields a *formally verified* design that exploits the robust nature of the feedback loops and allows some of the control messages to miss their deadlines, while still meeting the desired control performance. This goal is addressed from two sides (see Fig. 4.1). We first establish how frequently and in what order (or pattern) the control application can tolerate deadline violations. There, we obtain a LTL formulation that must be satisfied. We

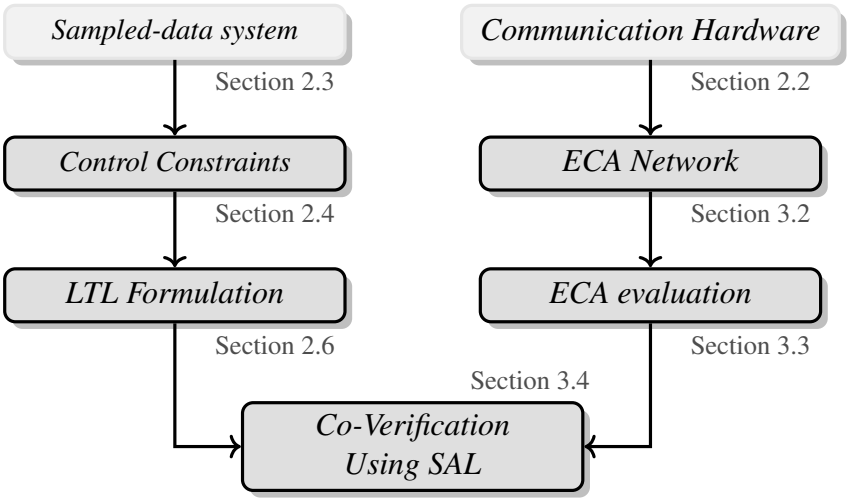


Figure 4.1: *Proposed co-verification framework*

then model the implementation architecture – i.e., the ECUs, the bus, and their schedules – so that it can be formally verified as a collection of automata, ECAs in our case. If the LTL formula is satisfied by the automaton at the end, the architecture in question is a valid implementation of the controller.

Implementation Architecture The CPS for this experiment is implemented on the platform implied by Fig. 4.2. The control application is partitioned in three tasks: sensor task (p_s), controller task (p_c), and actuator task (p_a) running on three different ECUs. ECU_1 and ECU_2 communicate over a shared bus, whereas ECU_3 is directly attached to ECU_2.

p_s reads state x_k in ECU_1 from the sensors and sends it to the bus. On ECU_2, p_c receives x_k, computes u_k based on x_k and stores it in a buffer. The periodically executed p_a then takes the computed input signal u_k – if it is present – and applies it at the beginning of the new period.

The corresponding ECA network is shown in Fig. 4.3. The stream we want to verify, i.e., the one we assume to be available for the discussed application, is the lowest-priority stream 6. The ECAs' update functions are chosen such that the streams remain separate. They are processed according to their priority on the bus and follow a time-triggered schedule on the TDMA-based ECU.

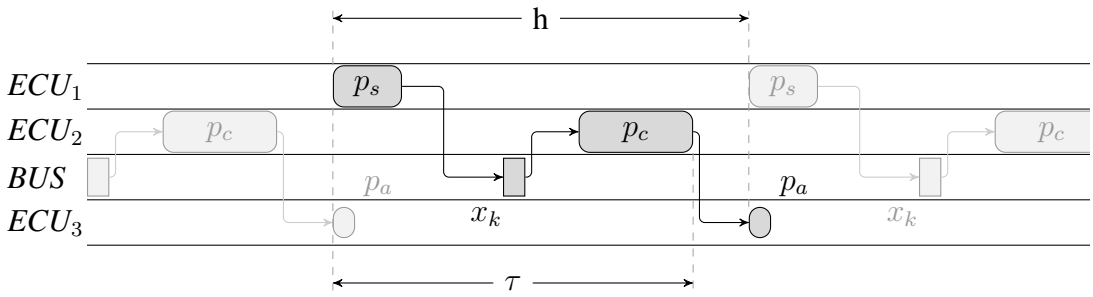


Figure 4.2: *Timing diagram of the distributed controller under consideration*

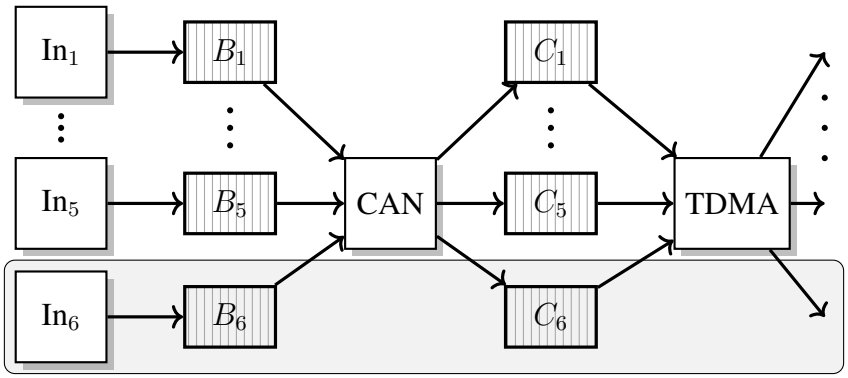


Figure 4.3: *Case study example modeled as ECA network*

Control strategy Consider a sampled-data system of the form introduced in Eq. (2.8).

$$x[k+1] = Ax[k] + B_1 u[k-1] \tag{4.1}$$

Here, a buffer is assumed to make control delay and thus deadline equal to sampling period, i.e., $d = h$. In this setting, $B_0 = 0$, meaning input $u[k]$ calculated from the current state cannot be applied in this period at all.

The time duration between reading sensor data x_k and the computation of u_k in ECU_2 is the varying transmission delay τ (see Fig. 4.2). Since the actuator can only apply it if the new signal is computed by the end of the period, the sampling period h represents a deadline for the control signal to reach ECU_3. If $\tau > h$, the application considers the message lost and must proceed accordingly. This is also referred to as deadline miss.

$$u_k = \begin{cases} Kx_{k-1} & \text{, if } \tau \leq h \\ 0 & \text{, if } \tau > h \end{cases}$$

The control input u_k as state-feedback with delayed state is thus applied only when the control message u_k meets its deadline and u_k is set to zero otherwise. This avoids energy expenditure in counter-productive directions. With this control strategy, the application becomes a switched system (2.21) with $A_c = A + BK$ and $A_o = A$ for the closed and open loop situation, respectively. The state must also be augmented to $x_{a,k} = \begin{bmatrix} x_k & x_{k-1} \end{bmatrix}'$ to account for the delay. See Eq. (2.11) for implementation details.

Experimental results Consider a second-order LTI system with

$$A = \begin{bmatrix} 0.611 & 0.284 \\ 0.239 & 0.7253 \end{bmatrix} \in \mathbb{R}^{2\times 2} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \in \mathbb{R}^{2\times 1}.$$

The sampling period that led to this discrete-time representation is $h = 3.5\,\mathrm{ms}$.

Controller gain $K = \begin{bmatrix} -0.1620 & -0.2200 \end{bmatrix}$ is designed with a pole placement approach that is unaware of faults but takes delay into account. A more involved control strategy that is custom-made for both delay and faults is presented in Section 4.2.

This system is expected to fulfill the exponential stability requirement from Eq. (2.24), *ExpStab*(l, ϵ) with $l = 5$ and $\epsilon = 0.7$. Recall that exponential stability requires a reduction by a factor of ϵ over l time steps. After evaluating the patterns as described in Section 2.6, it becomes evident that this property can be guaranteed by requiring the system to be both (2,3)-firm and (3,5)-firm with respect to its period. Plainly, this means that in any three samples at most one message, and in any 5 samples at most two messages can have delay $\tau > h$. As mentioned, the set of all acceptable patterns can thus be represented by a combination of (f,H)-firm deadlines where $H \leq l$.

Turning back to the architecture illustrated in Figs. 4.2 and 4.3, we recall that the discussed control application transmits messages over a shared communication bus. There are five other, higher-priority streams that interfere with these messages. All the messages are assumed to be periodic with jitter as detailed in Table 4.1. The control message has a period of $3.5\,\mathrm{ms}$ which is equal to the sampling period h of the discrete-time plant. As discussed, this period also constitutes the deadline for the messages in this case. The message length is assumed to be $8\,\mathrm{B}$ which results in $0.21\,\mathrm{ms}$ transmission time including overhead on a $512\,\mathrm{kbit}$ communication bus. The bus is considered to be preemptive. This does not only allow ignoring the complications described in Section 3.5, but also facilitates a comparison with other techniques. The execution times e_i for the TDMA ECU are given in the same table.

Relying on RTC, an algebraic, state-of-the-art timing analysis [184] is performed for the control messages in the given architecture. This yields a worst-case delay of $\tau = 4.84\,\mathrm{ms}$. Clearly, this is larger than the controller's sampling period of $T = 3.5\,\mathrm{ms}$ and the performance requirement could not be guaranteed by a design with such a worst-case delay.

Next, the ECA framework is applied to verify the derived (f,H)-firm deadlines as laid out in the overall co-verification approach (Fig. 4.1). ECAs are based on an implicitly chosen unit time step. The obvious choice would be the smallest time that occurs in the system, such as the bus arbitration time. This is very small however and leads to an explosion of the state space. In this case study, the ECA time unit is thus set to $0.5\,\mathrm{ms}$ such that the shared bus processes two messages in every step. With this period and the execution times e_i in mind, the streams are then scheduled sequentially on the TDMA ECU. In this setup, processing alternates between competing streams during the first $0.5\,\mathrm{ms}$ slot and the discussed stream during the other $0.5\,\mathrm{ms}$ slot.

After implementation in SAL (see Section 3.4), the LTL specification for the required (f,H)-firm deadlines is checked with the timestamping approach from Section 3.3.

Table 4.1: *Deadline verification case study parameters*

Task i	Period p_i	Jitter j_i	Execution time e_i
1	$1.0\,\mathrm{ms}$	$0.5\,\mathrm{ms}$	$50\,\mu\mathrm{s}$
2	$1.0\,\mathrm{ms}$	$1.0\,\mathrm{ms}$	$50\,\mu\mathrm{s}$
3	$2.0\,\mathrm{ms}$	$1.5\,\mathrm{ms}$	$125\,\mu\mathrm{s}$
4	$2.0\,\mathrm{ms}$	$0.5\,\mathrm{ms}$	$125\,\mu\mathrm{s}$
5	$3.0\,\mathrm{ms}$	$1.5\,\mathrm{ms}$	$150\,\mu\mathrm{s}$
6	$3.5\,\mathrm{ms}$	$0.5\,\mathrm{ms}$	$200\,\mu\mathrm{s}$

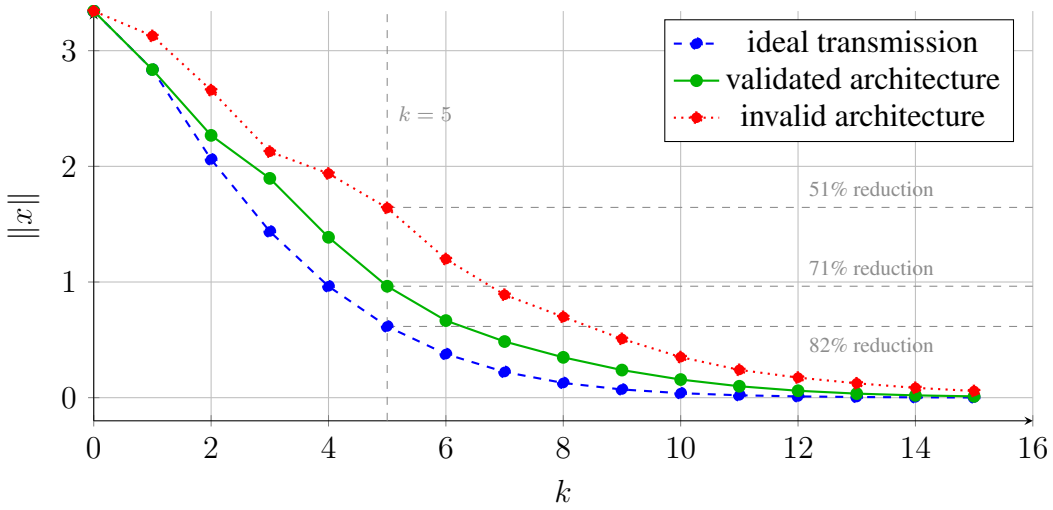


Figure 4.4: *Control performance evolution for different architectures*

Example of a valid architecture We consider the control specification of a (3,5) and (2,3)-firm deadline that, combined, guarantee the demanded exponential stability of the CPS. With the interfering message properties set as shown in Table 4.1 (load on the bus: $90\,\%$), the ECA framework validates these soft deadline requirements. This indicates that the control application can be implemented on the given architecture meeting the $ExpStab(l, \epsilon)$ requirement. To evaluate the resulting control performance, SAL delivers the worst-case trace of delay for the architecture with these message properties. The corresponding system behavior is then obtained from MATLAB via simulation. Fig. 4.4 compares the control performance from such a delay-trace and the control performance with ideal transmission without any deadline miss. It can be seen that the architecture that satisfies the specifications (that are derived from the exponential stability constraints) reduces an error signal by $71\,\%$ in 5 samples while the same system without any deadline miss (i.e., with ideal transmission) could reduce $82\,\%$ in the same duration.

Example of an invalid architecture The same control algorithm is also evaluated on a communication platform where the properties of the interfering messages differ slightly from Table 4.1. Consider changing j_4 from $j_4 = 0.5\,\text{ms}$ to $j_4 = 1.5\,\text{ms}$ or alternatively changing $p_5 = 3.0\,\text{ms}$ to $p_5 = 2.5\,\text{ms}$. Both changes are sufficient to make the resulting platform fail the (3,5) and (2,3)-firm deadline requirement. Consequently, this means that an architecture with modified message properties fails to guarantee the $ExpStab(5, 0.7)$ requirement.

We obtain a similar delay-trace from SAL as previously for the validated architecture for $p_5 = 2.5\,\text{ms}$ (load $92\,\%$). In Fig. 4.4, we can see that the control performance deviates significantly from the ideal case without deadline miss. Only $51\,\%$ error reduction is achieved when the CPS is implemented on top of this invalid architecture. This is less than the $71\,\%$ that the valid architecture achieved in 5 samples. In this example, the invalid platform still meets the controller's exponential stability criterion that any error signal should be reduced by at least $25\,\%$ in 5 samples. Keep in mind that the performance shown here is not the absolute worst

case. It is possible to find another delay-trace and initial condition through extensive simulation that will cause a violation of the control requirements.

The examples above demonstrate that a small change in the architecture (like period and jitter of one interfering message) can have a significant impact on the resulting control performance. This demonstrates the necessity for formal verification techniques in order to avoid the extensive testing and integration effort entailed by an incremental design process.

4.2 Fault-tolerant control design with delays under firm deadline assumption

In the previous sections ECAs serve to check and verify the performance of a distributed embedded control system closed over a faulty or severely constrained communication network. Such overloaded networks are common in cost-sensitive domains such as automotive. The assumption in the previous co-verification approach (Section 4.1) is that the controller is fully designed. Here, the situation is reversed and we aim to exploit robustness of the controller in another way to achieve a tighter design.

The communication platform is fixed a priori and the control law is considered changeable. In many cases this is a closer reflection of reality. The control law is usually software code that can be re-programmed without interfering with other applications. Changes in the communication hardware are virtually always prohibitively expensive; the feasible scheduling adjustments entail at least new analysis and testing of the overall system. For that purpose, this section treats the following research question: Given a bound on deadline misses, how to design a controller such that the desired stability and Quality of Control (QoC) requirements are met. The ECA framework can model a distributed embedded architecture and formally verify it according to property formulations of arbitrary complexity. With such a bound in mind, a fault-tolerant control strategy can adjust the input signal at runtime based on the observed occurrence of faults or message drops. The QoC under faulty communication improves significantly using such a fault-tolerant approach.

More precisely, this section introduces a LMI-based formulation for fault-tolerant control design that takes into account both the bound guaranteed by the model checking framework and delay. The resulting control strategy observes deadlines – that can be smaller than sampling period – at runtime and adjusts the linear gains with respect to violations. The constructive proof of Theorem 4.1 yields the existence of a Common Quadratic Lyapunov Function (CQLF) and the switching controller thus guarantees exponential stability relying on Theorem 2.6.

This new control design and implementation algorithm significantly improve the system's stability and reduced the required input over traditional methods such as ZOH.

Communication timing adjustments for DCC The overall challenge in this section is still the stabilization of a sampled-data system over a network. In order to employ Drop Compensation Control (DCC), the order of computation must be adjusted, however.

The difference to the general sampled-data system analysis from Section 2.3 lies in the sensing, communication and actuation timing. In addition to the total *sensor-to-actuator delay*

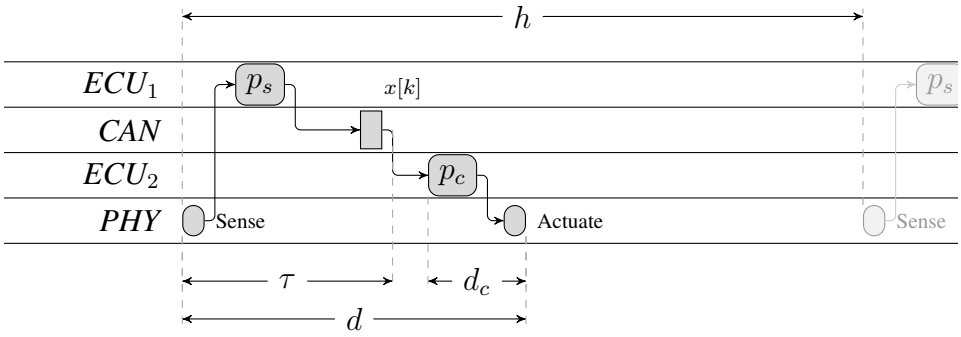


Figure 4.5: *The timing diagram for DCC differs from general sampled-data system analysis (Fig. 2.3). Controller task p_c requires constant time d_c that must be available after the decision about fault occurrence is taken. A fault thus occurs if variable delay $\tau > d - d_c$.*

d from Fig. 2.3, DCC must separately take into account the time required for computation. The computation phase is the default and often the only instant for adjustments to happen at runtime.

Fig. 4.5 displays the relevant timing pattern. The control system is divided into only two main tasks, sensor task p_s and controller task p_c. The duties of the previous actuator task p_a are integrated into p_c for simplicity. On ECU_1, state $x[k]$ is read from the sensor periodically. p_s further processes the reading and sends it out over the network to ECU_2 as a message. After the message is transmitted over the network, it is stored in the buffer of ECU_2. On ECU_2, p_c is triggered periodically. It checks whether a message has arrived in its buffer, calculates input signal $u[k]$ and adjusts the actuator accordingly.

τ is the variable time interval from sensor reading until the message arrives at the buffer of ECU_2. Controller task p_c then needs time d_c to calculate $u[k]$ and apply the input signal via the actuator. Sensor and actuator are triggered periodically with period h and a relative offset of d. Therefore, τ must not exceed $d - d_c$ for a successful actuation. A message is considered to be *dropped* when $\tau > (d - d_c)$. Under these circumstances, the continuous delayed system (2.4) can be considered a discrete-time sampled-data system with period h and constant sensor-to-actuator delay d. Recall the relevant dynamics from Eq. (2.8)

$$x[k+1] = Ax[k] + B_0 u[k] + B_1 u[k-1] \tag{4.2}$$

where the system matrices are calculated as laid out in Eq. (2.9). These matrices depend on control delay d and sampling period $h \geq d$.

The goal is to guarantee QoC in the form of exponential stability (Definition 2.3) under the presence of some dropped messages. However, package drops degrade QoC and potentially render the system unstable. Hence, messages cannot be dropped too frequently. The permissible amount of lost signals is quantified by the notion of (f, H)-firm deadline (Definition 2.7). For a given $(1, H+1)$-firmness, the control strategy is adapted online to ensure exponential stability.

Typically, the drops under consideration are caused by delay from contending messages and subsequent deadline violation as in Section 2.2. The fault tolerant control strategy can handle sporadic hardware faults or disturbance from outside the control setup in the same way. While it remains questionable whether such errors can be quantified in a safe way, fault-tolerant approaches are certainly preferable to standard designs if any mishaps are possible.

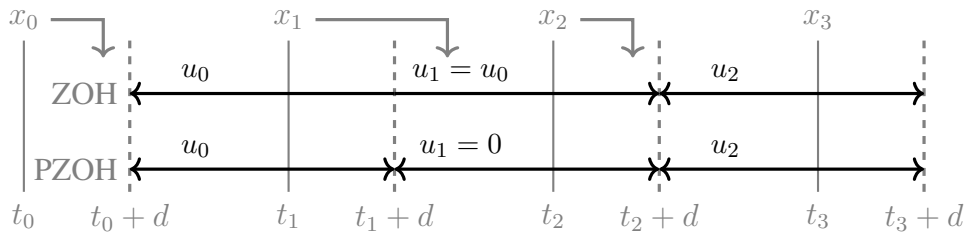


Figure 4.6: *Activation Sequences for ZOH and PZOH in reaction to two successful (x_0, x_2) and one failed (x_1) transmission. ZOH only changes the input after a successful transmission. PZOH sets the input to zero if a fault is detected.*

Actuation timing patterns The traditional implementation of a digital controller is a sample and hold block that stores the control signal until the periodically actuated controller finds a new one in its buffer. This is referred to as Zero-Order Hold (ZOH).

$$u[k] = \begin{cases} Kx[k] & \text{, if } \tau \leq d - d_c \\ u[k-1] & \text{, otherwise} \end{cases} \tag{4.3}$$

Note that certain works in the literature assume that ZOH is interrupt-driven and applies new signals as soon as they arrive. This thesis does not deal with the complicated analysis that the induced varying delays require and does not consider this interpretation therefore. An alternative approach periodically checks the buffer for fresh state information and only actuates if the last transmission was successful. This may in certain cases lead to higher input energy, but it avoids diverting the system into an undesirable direction. That approach is called Periodic Zero-Order Hold (PZOH) in this work.

$$u[k] = \begin{cases} Kx[k] & \text{, if } \tau \leq d - d_c \\ 0 & \text{, otherwise} \end{cases} \tag{4.4}$$

Fig. 4.6 illustrates the difference between these patterns.

DCC builds upon these patterns. In case of a miss, it adjusts the new input signal to lessen the impact of the old input signal. Its design is more involved than for ZOH, but it achieves superior QoC and does not introduce chattering.

Drop Compensation Control (DCC) design Any controller that is periodically checking its buffer for new information in PZOH fashion (see Fig. 4.6) can identify dropped messages at runtime. This motivates searching for a strategy that compensates for drops and thereby increases the robustness of the system. Fig. 4.7 shows the schematic implementation for such a drop controller. The drop controller counts the number of consecutive drops j, i.e., the number of instants where $\tau > d - d_c$, since the last successful update. In time step $k + j$, it then actuates the system based on $x[k-1]$, the last state that was transmitted, and $u[k-2]$, the input at that time.

$$u[k+j] = K_j x[k-1] + G_j u[k-2], \quad j = 0, 1, \ldots, m-1 \tag{4.5}$$

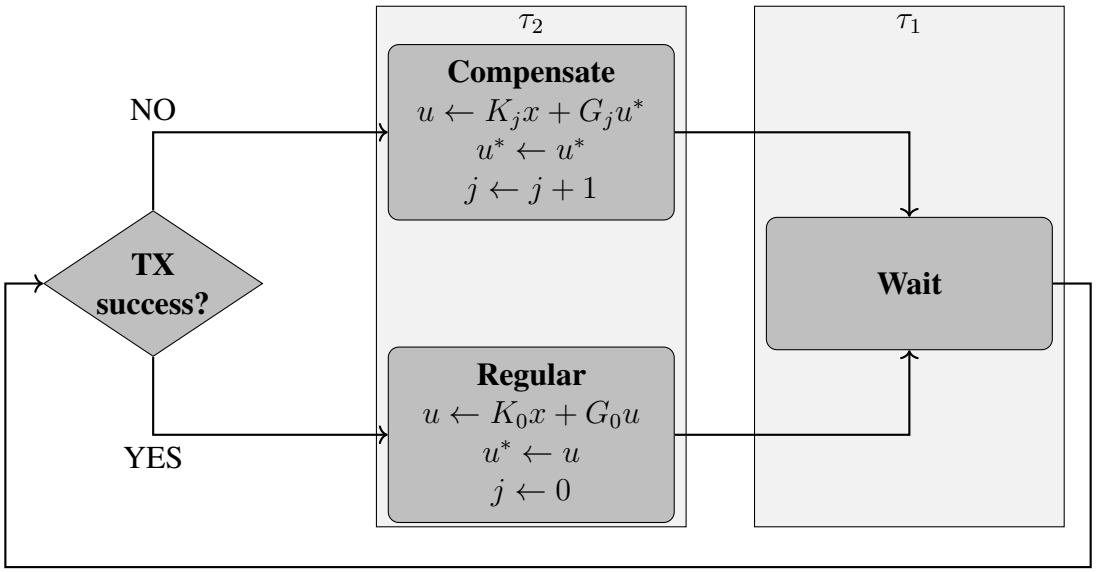


Figure 4.7: *Schematic representation of Drop Compensation Control (DCC). After a successful transmission (TX), the regular path is taken. If no new information is available, progressively higher compensation gains (K_j, G_j) are utilized.*

In the following, we discuss how to select gain matrices $(K_j, G_j)_{j=0,\dots,m}$ such that exponential stability of the system is guaranteed.

In the *nominal case*, there are no drops and the system is always actuated using (K_0, G_0). Using an extended state $X[k] := \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}'$ we can write this as

$$X[k+1] = \begin{bmatrix} A + B_0 K_0 & B_1 + B_0 G_0 \\ K_0 & G_0 \end{bmatrix} X[k] =: A_n X[k]. \tag{4.6}$$

In what follows, we develop an algorithm for designing the stabilizing controller in the *drop mode*. For $j = 1$ drops, it holds

$$\begin{aligned} x[k+1] =& Ax[k] + B_0 u[k] + B_1 u_{k-1} \\ =& A\big(Ax[k-1] + B_0 u[k-1] + B_1 u[k-2]\big) \\ &+ B_0\big(K_1 x[k-1] + G_1 u[k-2]\big) + B_1 u_{k-1}. \end{aligned} \tag{4.7}$$

Introducing $A_B := AB_0 + B_1$, this is equivalent to

$$X[k+1] = \underbrace{\begin{bmatrix} A^2 + A_B K_0 + B_0 K_1 & AB_1 + A_B G_0 + B_0 G_1 \\ K_1 & G_1 \end{bmatrix}}_{=:A_d^{(1)}} X[k-1].$$

For an arbitrary number of drops j, this construction can be iterated, resulting in the following closed loop dynamics.

$$X[k+1] = \begin{bmatrix} A_0^{(j)} & A_1^{(j)} \\ K_i & G_i \end{bmatrix} X[k-j] =: A_d^{(j)} X[k-j] \tag{4.8}$$

Here, the relevant system matrices $A_0^{(j)}$ and $A_1^{(j)}$ are defined by

$$A_0^{(j)} := A^{j+1} + \sum_{\ell=1}^{j} A^{j-\ell} A_B K_{\ell-1} + B_0 K_j$$
$$A_1^{(j)} := A^i B_1 + \sum_{\ell=1}^{j-1} A^{j-\ell} A_B G_{\ell-1} + B_0 G_j. \tag{4.9}$$

The goal remains designing a controller that renders the switching system in (4.8) exponentially stable (Definition 2.3) given that there is a bound on the consecutive drops, i.e., $j \leq H$. The following theorem demonstrates how to find such a controller using LMI.

Theorem 4.1 (Drop compensation controller)**.** *Suppose the communication of the system meets a $(1, H+1)$-firm deadline requirement, i.e., there are at most H consecutive drops. If there exist matrices $E_j \in \mathbb{R}^{p\times n}$, $F_j \in \mathbb{R}^{p\times p}$, $Q_1 \in \mathbb{R}^{n\times n}$, $Q_2 \in \mathbb{R}^{p\times p}$ with $Q_1, Q_2 \succ 0$, and positive scalar $\gamma < 1$ such that LMI* (4.11) *and the system of LMIs* (4.12) *are satisfied. Then, utilizing $K_j \in \mathbb{R}^{p\times n}$ and $G_j \in \mathbb{R}^{p\times p}$ as given by* (4.10) *in control strategy* (4.5) *guarantees exponential stability of system* (4.2)*.*

$$K_j = E_j Q_1^{-1} \qquad G_j = F_j Q_2^{-1} \tag{4.10}$$

$$\begin{bmatrix} -\gamma Q_1 & * & * & * \\ \mathbf{0} & -\gamma Q_2 & * & * \\ AQ_1 + B_0E_0 & B_1Q_2 + B_0F_0 & -Q_1 & * \\ E_0 & F_0 & \mathbf{0} & -Q_2 \end{bmatrix} \prec \mathbf{0} \tag{4.11}$$

$$\begin{bmatrix} -Q & * \\ L_j & -Q \end{bmatrix} \prec \mathbf{0}, \;\; j = 1, \cdots, H \tag{4.12}$$

Here $\star$ refers to symmetric completion of the matrix and L_j is given as follows.

$$L_j := \begin{bmatrix} A^{j+1}Q_1 + \sum_{\ell=1}^{j} A^{j-\ell} A_B E_{\ell-1} + B_0 E_j & A^i B_1 Q_2 + \sum_{\ell=1}^{j-1} A^{j-\ell} A_B F_{\ell-1} + B_0 F_j \\ E_j & F_j \end{bmatrix}$$

Proof. Using the Schur complement, LMI (4.11) can be rewritten to

$$L_0' Q^{-1} L_0 - \gamma Q \prec \mathbf{0} \tag{4.13}$$

where matrices Q, L_0 are defined as

$$Q = \begin{bmatrix} Q_1 & \mathbf{0} \\ \mathbf{0} & Q_2 \end{bmatrix} \qquad L_0 = \begin{bmatrix} AQ_1 + B_0E_0 & B_1Q_2 + B_0F_0 \\ E_0 & F_0 \end{bmatrix}$$

Substituting (4.10) and keeping nominal system matrix A_n from (4.6) in mind, we obtain

$$L_0 = \begin{bmatrix} AQ_1 + B_0K_0Q_1 & B_1Q_2 + B_0G_0Q_2 \\ K_0Q_1 & G_0Q_2 \end{bmatrix} = A_n \cdot Q. \tag{4.14}$$

With this, (4.13) can be further reformulated to

$$QA_n'Q^{-1}A_nQ - \gamma Q \prec \mathbf{0}. \tag{4.15}$$

Multiplying (4.15) from left and right by Q^{-1}, and defining a new positive definite matrix $P := Q^{-1}$, we arrive at the following inequality.

$$A_n'PA_n - \gamma P \prec \mathbf{0} \tag{4.16}$$

This proves that (4.11) guarantees the existence of a Lyapunov function for the nominal case.

In a similar fashion to the nominal case, it can be shown for the drop mode that inequality (4.12) implies

$$A_d^{(j)\prime}PA_d^{(j)} - P \prec \mathbf{0}, \quad j = 1, \cdots, m \tag{4.17}$$

Together, these inequalities imply that a CQLF $V(X) = X'PX$ exists for systems (4.6) and (4.8). Recalling Theorem 2.6, this completes the proof. □

Remark 4.2 (Relation of drop compensation and state estimation)**.** *One can view DCC as a special case of state estimation scheme.*

Fault-tolerant control case study In order to compare DCC to the established ZOH design, consider the following continuous-time LTI plant with highly unstable system matrix $A^{(c)}$ and two variations of input matrices $B_1^{(c)}$, $B_2^{(c)}$ (consider (2.4) for reference).

$$A^{(c)} = \begin{bmatrix} 0 & 1 \\ 6.31 & -15.48 \end{bmatrix} \qquad B_1^{(c)} = \begin{bmatrix} 0 \\ 10 \end{bmatrix} \qquad B_2^{(c)} = \begin{bmatrix} 0 \\ 1000 \end{bmatrix} \tag{4.18}$$

To bring this into the required discrete-time form (4.2), a sampling period of $h = 10\,\mathrm{ms}$ and a delay of $d = 4\,\mathrm{ms}$ is chosen. A, B_0, B_1 are then calculated according to (2.9). The system is required to maintain exponential stability tolerating a maximum of $m = 2$ consecutive drops. Theorem 4.1 yields the desired control gains K_i, G_i for this goal.

Fig. 4.8 shows the behavior of the plant with the various control strategies subject to random drops within the specifications. All traces start with an initial disturbance of $x_0 = \begin{bmatrix} 0.25 & 0 \end{bmatrix}'$. On the left, we see that PZOH is clearly not suited for the bigger input amplitude as it introduces considerable chatter in u and converges much slower towards 0 in x_1 than the other approaches. In case of a smaller input amplitude, as with $B_2^{(c)}$ on the right, it is a more reasonable choice than ZOH however. Since this system is more sensitive, ZOH can easily lead to over-actuating and hence instability there. The proposed DCC strategy does not suffer from either of these drawbacks and performs well for both input matrices.

Formal verification The described control application is implemented as the last of five exemplary streams with periods P and jitter j as follows.

$$\begin{aligned} P &= \begin{bmatrix} 4\,\mathrm{ms} & 5\,\mathrm{ms} & 7.5\,\mathrm{ms} & 7.5\,\mathrm{ms} & 10\,\mathrm{ms} \end{bmatrix} \\ j &= \begin{bmatrix} 1\,\mathrm{ms} & 1.5\,\mathrm{ms} & 1.0\,\mathrm{ms} & 1.5\,\mathrm{ms} & 0.5\,\mathrm{ms} \end{bmatrix} \end{aligned} \tag{4.19}$$

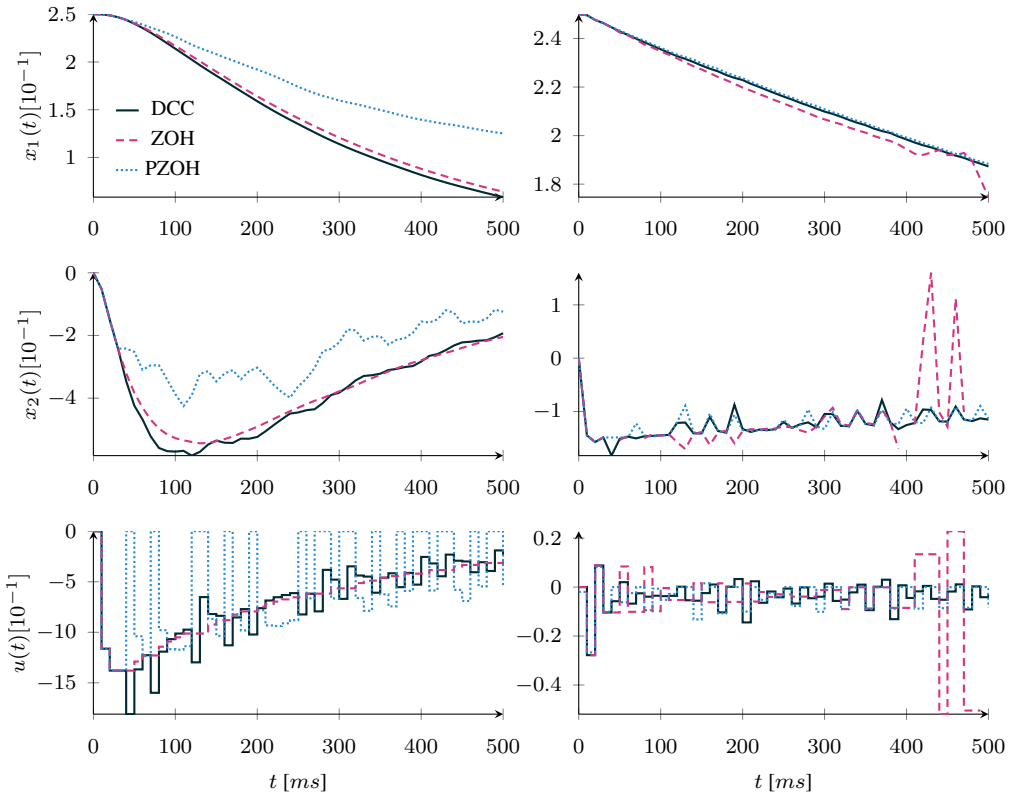


Figure 4.8: *Comparing x_1, x_2, u (top to bottom) for system* (4.18) *with matrix $B_1^{(c)}$ (left) and $B_2^{(c)}$ (right): ZOH (dashed) performs well for large inputs as required by $B_1^{(c)}$; PZOH (dotted) is well-suited for small inputs as induced by $B_2^{(c)}$; DCC (solid) remains stable and more steady in both situations.*

The streams are traveling over a network with two service automatons. The first is a TDMA automaton that rotates 10 slots, each $0.5\,\mathrm{ms}$ long. In its second slot, it processes one item of stream #5, the stream of interest. The sixth slot is reserved for maintenance tasks and the remaining slots each process higher priority messages.

After the TDMA ECU, messages are further processed by a bus with speed-stepping that schedules according to a synchronized FPNS scheme. It starts out processing 1 message in every second slot. If there are more than 2 messages waiting to be processed in total, it increases its speed to one message per slot. If there are no more messages waiting, it returns to processing 1 message in every second slot. Messages are arbitrated at the beginning of the slot, i.e., the ECA processes the highest priority message that was in the buffer at the end of the last slot.

Verification Result and Combined Design If the system runs purely on the starting speed, stream #5 will only succeed occasionally. If it always utilizes the high speed, there are no violations for a deadline of $d - d_c = 3\,\mathrm{ms}$. Under the described speed stepping scheme, the model checking implementation finds two design points. It is guaranteed that there will be at most five consecutive misses for a deadline $d - d_c = 3\,\mathrm{ms}$ and at most two for a deadline $d - d_c = 4\,\mathrm{ms}$. With controllers designed at both points for $A^{(c)}$, $B_2^{(c)}$ from (4.18), we evaluate

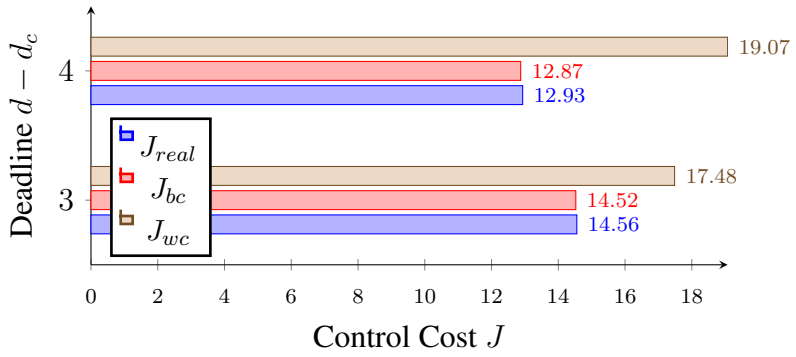


Figure 4.9: *Comparing two DCC design points found by the ECA network based verification. Designing for higher delay, but less drops leads to a lower cost in the best (J_b) and the simulated case (J_{real}), only the worst case (J_{wc}) is inferior.*

the performance using a typical quadratic cost-term punishing both non-augmented state and input. Such costs and the corresponding conversion from continuous- to discrete-time domain have been described in the lead-up to Eq.(2.19). The cost parameters $Q = I_n, R = I_p$ are chosen to be identity matrices of appropriate size.

We examine data from $N = 300$ randomly initialized Monte-Carlo simulations of length $T_{\max} = 5400\,\mathrm{ms}$. The comparison is shown in Fig. 4.9. Here, J_{wc} refers to the worst case where the maximum number of drops always occur and J_{bc} refers to the best case where no drops occur. In addition to the verification, the system is also modeled using a SYSTEMC implementation. This generates a trace of the communication that is subsequently fed into the MATLAB simulation of the control system. The costs that are observed from that simulation are the realistic costs J_{real}.

Designing for longer delay with fewer losses clearly outperforms designs that accept more drops under a stricter deadline. Note however, that increasing the delay to the point where no more drops occur often renders the corresponding LMI system infeasible for the solver.

Comparison with optimistic lossless design Depending on the design flow, the designer may have more flexibility with regards to sampling period h in certain cases. It is thus of interest to see how the proposed design would fare compared to another implementation with longer sampling period, but without losses. Fig. 4.10 shows the optimistic pattern assumed for this comparison. Given a (1, H+1)-firm deadline, it forgoes the transmission of the two potentially failing messages. This yields a lossless pattern with sampling period $h_{ll} = (1 + H) \cdot h$ with no changes in deadline $d - d_c$. This is *optimistic* and only serves to demonstrate the value of the control design because the drop pattern is typically irregular and unknown a priori.

For this comparison, consider again the system from (4.18) with $B_2^{(c)}$ and set $d - d_c = 2\,\mathrm{ms}$, $H = 2$. The performance is evaluated using the same quadratic cost-term with $Q = I_n, R = I_p$. Simulation length and cost per step have been adjusted to account for different sampling periods across the various examples and the longer slots in the lossless cases. Note that these results are therefore not comparable with Fig. 4.9.

Fig. 4.11 summarizes the result of the comparison of DCC with a lossless system for sam-

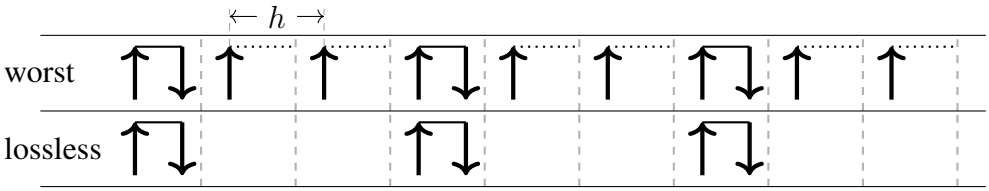


Figure 4.10: *Assuming the message drops follow the regular worst-case pattern under sampling period h, we could also regulate the system using the larger period $h_{ll} = 3{\cdot}h$. Note that this pattern is introduced to illustrate the value of our control design and that in real systems, it is actually not guaranteed to be lossless.*

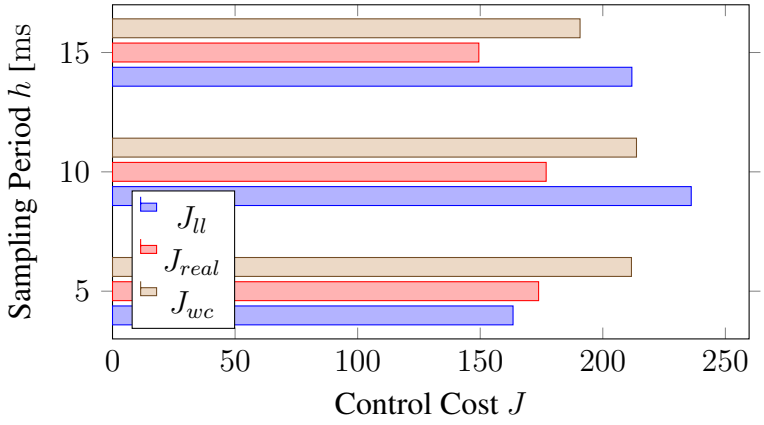


Figure 4.11: *Comparing the fault-tolerant controller (J_{real} and J_{wc}) to the lossless design (J_{ll}) with $h_{ll} = 3 \cdot h$ from Fig. 4.10: For $h = 5$, J_{real} and J_{ll} are on par, indicating oversampling; For larger h, J_{real} is significantly smaller than J_{ll} showing the value of the combination design.*

pling periods $h_1 = 5\,\text{ms}$, $h_2 = 10\,\text{ms}$, $h_3 = 15\,\text{ms}$. This data stems from Monte-Carlo simulation with 500 random initializations of x_0 and 150 time steps afterwards. For the smallest sampling period the system is oversampled since reducing the sampling frequency does not reduce QoC even when compared to J_{bc} the best (or lossless) case for the smaller period h (not shown). The lossless design outperforms DCC here, albeit only slightly when compared to the real performance. Once a critical sampling period is reached, the system becomes sensitive to further sampling reductions. DCC designs become beneficial in such circumstances. Here, even the worst case pattern for the faster sampling achieves $10\,\%$ better QoC; the real case, as obtained by simulating, is about $25\,\%$ superior.

4.3 Related work

The contributions of this section serve several purposes and as such relate with various fields. Both the guaranteed performance approach from Section 4.1 and the fault-tolerant controller from Section 4.2 help operating under reduced communication. From this perspective, they relate to techniques with the same goal, like event-triggered control. They also achieve a performance guarantee, giving them a relationship to formal verification of CPSs. Fault-tolerant

control is still relevant on its own without talking about errors from communication and there have been many contributions in that area. Finally, treatment of delay is also found in the area of Networked Control System (NCS). Overall, this section presents several small contributions in all these areas. Mainly, it demonstrates how to bring these fields together and make a first step towards co-verification and co-design of control system and hardware platform. The necessary literature to understand the underlying ECA framework is presented in Section 3.8.

Networked Control Systems (NCSs) The NCS paradigm essentially builds a theory around the control engineering perspective of the interaction between the controller and the communication platform (see Fig. 2.2). It is typically assumed that the knowledge of the communication platform is limited and there is no control over it. The network is often modeled with random behavior or even considered an opponent who may act in the worst possible fashion (subject to mathematically convenient fairness constraints).

Hespanha provides an overview of the NCS area in [77]. Detailing the most relevant findings, this work firstly treats optimal estimation of discrete-time LTI systems over lossy networks. Assuming Bernoulli dropouts a Kalman filter approach remains optimal until it breaks down once the dropout probability reaches a critical value [163]. Smart sensors that compute state estimates locally and then transmit them dramatically improve the robustness against drops compared to dumb sensors that directly transmit their readings [191].

The next important topic in NCS is stability under delay. In [182], Walsh derives a bound for the maximum allowable transfer interval below which it is guaranteed that a linear NCS remains globally exponentially stable. Branicky and Zhang take a different approach in [30] and [197]. They model the NCS as hybrid system and evaluate the Schur-ness of a matrix containing various system matrices. In [28], they further examine the effects of dropouts. For a stable closed-loop system they develop bounds for the dropout rate based on the eigenvalues of the closed- and open-loop matrices.

Suh goes one step further in [171] where he considers stability under non-uniform sampling. Toward this, he first finds norm sampling periods for which the norm bound of the system is small. These subsequently help form an LMI that guarantees the overall stability. A similar result has been simultaneously reported by Fujioka in [68]. The main advantage of the latter is "the development of a concrete algorithm for the state feedback synthesis with a guarantee of convergence".

The delay introduced in a NCS also motivates adjusting established control strategies. In [196], Zhang shows a new approach for designing the LQG gain under delay, i.e., the control law that minimizes the quadratic costs from (2.19). The authors' method is computationally cheaper than the naive approach of augmenting the state space as in (2.11) and only then computing the gain for the larger matrix. They mention however that this approach still works as long as only a single delay is involved and the matrix does not become too large.

Model Predictive Control (MPC) has also been adjusted for NCS. Soudbakhsh presents an MPC formulation with delay in [164]. While the paper focuses on the parallel implementation for faster calculation, it begins with a derivation of the augmented LTI discrete-time matrices for taking delay into account. In [135], a controller is implemented over a platform that has two communication channels with different delay. After augmenting the state space, the authors solve a mixed integer quadratic program at runtime to decide switching between the channels.

Mixed integer programs are computationally hard to solve, however. While the approach may hence not provide benefits in practice, it is of interest as reference solution for future work in this direction. In fact, comparing it with Voit's approach from [180] would be interesting. That scheme also switches between two channels with different delay. The authors use an adaptive controller that guarantees bounded behavior as long as impulse disturbances have sufficient inter-arrival time. This requires no optimization at runtime and is thus more suitable for embedded environments.

Another set of contributions is concerned with simulations of NCS. Cervin's JITTERBUG addresses this aspect [115]. In the words of the author, it " is a MATLAB-based toolbox that is used to analyze linear control systems with time-varying delays" [38]. The main downside is that delays in the simulation follow standard random assumptions like independence that may not hold in practice. Nevertheless, this tool provides a valuable ad-hoc gauge for QoC under delay and drops.

All NCS studies mentioned here have in common that they use tools from control theory to mitigate effects from the communication they consider unavoidable. Interestingly, it is extremely rare that data distortion is considered in this context. The design of that architecture which is after all an entirely human creation as well has only been considered in more recent co-design approaches.

Event-triggered control Going beyond the NCS goal of ensuring stability under adverse communication but still working in the same platform-agnostic settings, event-triggered control aims to reduce communication and communication in some sense.

The first family of event-triggered control works survey a performance metric. They trigger when the system comes dangerously close to failing the performance requirement. Differences between them must be searched in the assumptions they make about actuating without computation, the performance metric, etc.

[172] is a prominent example that observes the decrease of a Lyapunov function and triggers the system when the descent is no longer fast enough. Using a Lyapunov function in this way even allows treating nonlinear systems. The authors provide an accompanying schedulability criterion and prove that this scheme leads to inter-arrival times that are bounded below. In other words, it cannot happen that computation and actuator task are triggered infinitely often. It remains an open question how they measure the state in continuous fashion but with low computation requirements. If an observer is involved to derive the state from a measurement, this appears challenging.

A similar scheme based on Lyapunov functions is pursued by the same group in [128]. There, instead of reducing the sampling per se, the goal is less network communication. In [129] they aim for decreased listening times. They claim "it is a well-known phenomena [sic!] in the sensor networks community that reducing listening times has a bigger impact on the power burden than reducing transmissions", citing [193]. The paper begins by looking for a stabilizing sample & hold implementation such that every state can be updated independently. The authors then design a simple triggering law and prove that it is effective. The assumption appears to be that the controller mainly deals with sudden errors, not ongoing disturbance.

[8] describes a similar approach: self-triggering. The idea here is that the controller decides its next invocation time itself during execution. In this way, ongoing measurements of the state

are no longer necessary. The authors define an operating region and then design a Lyapunov function using a sum-of-squares technique. This subsequently serves to form the predictive triggering law, as before. [155] moves in the same direction but with more focus on scheduling the self-triggered tasks. The authors contribute a heuristic based on cost-function approximations. They ensure stability through design-time verification and by construction.

A major downside of all event-triggering approaches that rely on a Lyapunov function for measuring progress is that they ignore input costs. Since the Lyapunov function is only concerned with the state evolution, it may be perfectly acceptable for them to employ more energy in an electric motor to correct a larger deviation caused by a few missed, or saved transmissions. If higher input energy is unacceptable, the following cost-based scheme may be an option. In [190], the transmissions and the estimation error cost in a remote observer problem are weighted and minimized together. The authors show that this corresponds to a dynamic programming problem. Molin later shows in [131] and [132] that this scheme is also useful for event-triggered control. In particular, the optimal controller can be designed separately and thus remains the LQG regulator gain. Dynamic programming is applied subsequently to find the optimal triggering law. The downside of dynamic programming is that it requires a discretization of the state space. Such a grid grows exponentially in the dimension of the state space and hence usually limits the applicability of the approach to dimensions smaller than 4. As a remedy, there are many approximate approaches for the calculation of triggering laws. [112], for instance, uses a sum-of-squares approach that works well for systems up to dimension 8.

An earlier event-triggered control approach simply overlays a feedback controller for the task periods on top of an actual control system implementation. This was discussed first by Eker in [61]. There, if the measured cost term is far from the reference value, sampling is increased. Henriksson extends this work in [75] by considering feedback from the plant states in addition to the linear quadratic cost function. Furthermore, he also models delay. In [37], the group considers more general controller designs. Whereas their works were limited to LQG before, they now treat pole placement as well. This paper works with sets of sampling periods, however. This allows building lookup tables, but is somewhat less flexible than the earlier works. These works consider an additive disturbance term as in (2.3). This is possible because they optimize the expected value of the costs and do not consider the worst case performance.

Zhang proposes adjusting the sampling period according to noise in the system as measured by the H_∞ norm in [195]. The authors prove that the corresponding cost function is monotonic and thus apply a gradient descent algorithm to determine the sampling periods offline. Learning from that method's results, they propose a heuristic for online scheduling.

Finally, the main issue of all event-triggered schemes and perhaps the reason why they have not found widespread application so far, is their difficult integration in real-time systems. The proposed schemes may provide identical performance with reduced communication *on average*, but they expect to receive computation and actuation when they request it. When provisioning for the worst case, it is thus impossible to downsize the system even though the average requirements went down after adopting event-triggering.

Cyber-physical System (CPS) design Whereas the computation and communication platform is a randomly behaving adversary for NCS researchers, the CPS paradigm takes that hardware into account with more detail. Lee argues in his position paper [108] that today's comput-

ing and networking abstractions do not match the inherent concurrency and relentless passage of time that the physical world exhibits. Real-time guarantees would benefit many industries, from transportation systems to financial networks. However, "lack of temporal semantics and adequate concurrency models in computing, and today's 'best effort' networking technologies make predictable and reliable real-time performance difficult, at best". In a nutshell, he points out that in spite of improvements according to Moore's Law for general computing the performance in real-time computing has practically stood still during the last 20 years. According to him, we need new semantics to change that.

In [109], Lee expands upon his initial paper and calls for robustness on all abstraction levels. This work also contains a vision of how timing guarantees can be achieved by including them into the Application Programming Interface (API) across all layers.

The main contribution of Lee's group in this direction is the PTOLEMY project. In the main reference [62], the team explains how they integrate the many heterogeneous subsystems into one consistent framework. They avoid amorphous heterogeneity where actors interface in various ways because it leads to unexpected system behaviors. Instead, they propose hierarchical heterogeneity. There, they require that the model of computation includes the flow of data and remains compositional after aggregation. After associating a number of nodes with each other, this group acts as new node and can participate in further composition. The tool itself provides a number of such models that can be composed in a Graphical User Interface (GUI). An example from their paper is the control system from Fig. 2.2. It has an ODE model for the plant that interacts with a discrete-time controller via sampling. In addition to modeling and simulation, PTOLEMY also provides a code generator.

Formal verification of control systems Formal verification of high-level controller models and, to some extent, also the verification of the generated code has been studied before.

Reachability analysis is a popular discipline, for instance. Its adherents are concerned with finding safe enclosures for nonlinear ODE systems like (2.1). Often, the contributions are limited to one of the simpler subclasses, however. Althoff's CORA [3], for instance, uses zonotopes to calculate the reachable set of both linear and nonlinear systems. Frehse's SPACEEX [66] focuses on LTI systems and can thus achieve tighter approximations and higher efficiency. It appears to be the only tool in this domain that handles systems with state dimension over 20.

The goal in reachability analysis is always guaranteeing that a given system does not reach a specified "bad" area of the state space. Game-theoretic control design goes even further and searches a control law that guarantees such properties by construction. Examples like TULIP [188] and PESSOA [130] both begin by discretizing the state space. Next, they employ model-checking and game-theory techniques to find the control strategy. The strategy itself is represented by the values it takes at the points of the discretization, akin to a large non-uniform lookup table.

None of these approaches has the communication platform in mind. Their analysis starts and ends with the control engineering perspective, i.e., the dynamic system alone. The control software and the code itself have been investigated more recently as well. COSTAN from [7], for instance, derives bounds on implementation errors that can be tolerated and then checks if the maximal error in the controller software respects the tolerance threshold. [64] also deals with static analysis of the C code that implements control functionality. This work combines

Lyapunov stability and Hoare logic. The authors suggest annotating MATLAB source code with *pre* and *post* expressions that are assumed to hold before and after a statement, respectively. These annotations then translate to the generated C code and can be verified independently by a proof checker.

Co-verification approaches The previous passage contains many formal approaches for control systems. None of them includes the computation and communication hardware as well as its timing properties into their framework, however. Both reachability analysis and the discretization approaches are already exhausting contemporary off-the-shelf computers. It is unlikely that they can simply be extended with knowledge about the architecture. As implementation architectures become more complex and distributed, however, the semantic gap between high-level controller models and their actual implementations continues to increase, necessitating such co-verification.

Alur and Weiss have proposed Büchi automaton models for various CPS performance specifications in [5] and [185]. These are also discussed in Section 2.6. Originally intended for online scheduling of processor tasks, they also serve as verifiable properties. Building on their work, [91] describes how such a Büchi automaton specification can be used in conjunction with the ECA framework. By constructing an ECA model of the communication platform the overall performance of a CPS can be guaranteed by verifying the corresponding specification in a model checker. This paper has been reproduced in Section 4.1. The idea is similar to that in [101] where model checking also jointly verifies controllers and their implementation platforms. The difference is two-fold. [101] relies on the RTC-UPPAAL interface proposed in [104]. The advantages and drawbacks of ECAs compared to TAs, as discussed in Section 3.8, hence apply here as well. In addition to a more explicit model of the architecture, Section 4.1 also utilizes a more general interface for the performance specifications.

Following [91], a later technique for drop analysis relying on RTC directly is developed in [165]. This approach for generating (f,H)-firm deadline guarantees is highly efficient, but it cannot deal with state dependency. Please refer to Section 3.8 for these limitations and the original motivation behind the ECA framework.

Co-design techniques Instead of verifying a certain platform behavior a posteriori, co-design techniques include QoC metrics during the platform composition. There are not many contributions in this space and they are not strictly superior to co-verification methods. Often, they optimize a cost function instead of guaranteeing a certain performance. In other cases, they determine costs via simulation or the architecture they work with is more favorable.

Schneider solves a Constraint Satisfaction Problem (CSP) in [158] to find a feasible FlexRay schedule. FlexRay is an upcoming bus technology that offers a static (time-triggered) and a dynamic (priority-based) segment. Statically scheduled messages exhibit predictable behavior, but there are limited slots to distribute. For each control application, this work determines a suitable period from the discrete set FlexRay offers by simulation. The applications are then synthesized into a single schedule by solving the CSP.

[119] pursues another interesting idea for schedule synthesis in time-triggered architectures. It uses evolutionary algorithms to determine the best periods and corresponding allocation. Such

an optimizer generates candidates from a given gene pool, here the available FlexRay slots. It then evaluates the quality of the resulting feedback loops by creating the LQG regulator gain according to the implied period and simulating. Over many cycles where good candidates are treated favorably in some sense, the algorithm finds a good result. These techniques are not guaranteed to find the global or even a local minimum. They do work well for many hard problems in practice, however. Here, they manage to find a non-equidistant scheduling sequence that clearly improves QoC compared to a standard round-robin approach. Interestingly, instead of scheduling 3 systems in the order "123123", the technique often finds irregular cases like "123231". A human designer would arguably not explore every such ordering and it is somewhat surprising that they can be beneficial.

Fault-tolerant control design Section 4.1 presents the Drop Compensation Control (DCC) approach from [90]. This paper is inspired by the aforementioned [165]. Besides the discussed technique for determining message drop patterns, it also contains an LMI-based test for exponential stability. No control law is designed there, but this test motivates the controller synthesis in [90]. It is the first synthesis based on the (f,H)-firm interface backed by ECA verification. A similar LMI-based design approach is also presented in [194]. It does, however, assume zero sensor-to-actuator delay and does not justify the fault bound it relies on.

[76] introduces patterns similar to ZOH, PZOH from a NCS perspective. This work also contributes a predictive outage control technique. This approach is based on predictive form and implemented using an additional computational block next to the actuator to compute new input signals in case no new messages arrived. The input signal there is computed using the previous inputs of the system. This means it requires more computation than the approach from [90].

Other contributions in the fault-tolerant control area are less closely related, but still relevant. In [29], Branicky deals with a system switching between several subsystems that all have their individual Lyapunov functions, ensuring their stability. He shows that the overall system nevertheless need not be stable. He then introduces limitations on the switching that ensure stability in such a mixed situation.

5 Quantitative Models for Charge Transfers in Active Cell Balancing (ACB)

The recurring theme of this thesis is the interaction between digital and physical. Such interaction commonly occurs in digital control systems where a controller periodically samples a physical process to subsequently compute a new input signal. In the previous chapters, we have seen that this interaction becomes significant when the timing of the underlying hardware platform is no longer negligible compared to the timing of the physical process. This interaction does also become important, however, when the actuation of the physical process is inherently digital. Switched systems, for instance, only have a discrete set of inputs that can be selected. For the linear, discrete-time case, where all possible switching instants are assumed to be predefined, these systems have been discussed in Section 2.5. In other cases, the system undergoes a fixed sequence of configurations and the timing of the corresponding transitions becomes the fundamental input variable. A field where such systems arise is the increasingly important area of Active Cell Balancing (ACB) that we investigate in this chapter. In ACB, internal charge transfers are performed, often at kilohertz frequencies, to improve the performance of battery packs.

The goal of this chapter are quantitative models for component selection, performance evaluation, and strategy design in the later stages of the ACB design flow described in Section 1.2. To that end, we first revisit the motivation for ACB, position this thesis with regard to the overall ACB design flow, and present the main challenges (Section 5.1). We then limit the analysis to inductor-based architectures and introduce their charge transfer mechanism (Section 5.2).

The remainder of this chapter develops accurate, yet computationally efficient models for large-scale simulations (Section 5.8); optimization is the subject of Chapter 6. For these simulation models, the transfer dynamics are localized using an equivalent circuit abstraction (Section 5.3) to yield an ODE where any battery model can be inserted (Section 5.4). After describing high-level actuation interfaces that make long-term transfers more manageable (Sec-

tion 5.5), we go over additional operating losses from the transistor switching that must be considered, e.g., when transferring at low current (Section 5.6). Even though a numerical solver can now calculate the current behavior within a phase directly, the high computation times motivate further reformulations to obtain closed-form expressions (Section 5.7). These help implement the actuation interfaces and are a requirement for rapid simulation. After presenting the faster simulation models (Section 5.8), the chapter concludes with a discussion on related work and other implementations of ACB (Section 5.9).

5.1 ACB: Motivation, design flow, and challenges

This section presents the motivation for balancing and subsequently covers the corresponding design goals and challenges. It also describes the four phases of the ACB design flow and how this thesis relates to them.

Motivation Balancing becomes necessary in most Electrical Energy Storage (EES) applications since Li-Ion battery cells do not behave identically. Justified by their energy density, these cells currently form the basis for a wide range of applications, like smartphones, laptops, or EVs. Their cell chemistry limits voltage to about $4\mathrm{V}$ and many cells are hence typically connected in series and/or parallel to form a battery pack that provides the required power output. In such setups, imbalances between the cells arise because cooling is non-homogeneous and because cell parameters exhibit significant variation, both initially and increasingly over time.

Gogoana [72], for instance, measures up to 25 % difference in internal resistance and 3.5 % difference in capacity over 72 fresh, commercially available $2.2\,\mathrm{A\,h}$ $LiFePO_4$ cells. Similarly, Dubarry [58] finds up to 3.8 % ($\pm$ 1.9 %) deviation in capacity and 60 % ($\pm$ 30 %) in internal resistance for a lot of $300\,\mathrm{mA\,h}$ $LiCoO_2$ cells. While such differences can be alleviated by clustering cells, e.g., by weight [161], the issue also grows over time as cells age. In [19], 48 cells with $1.85\,\mathrm{A\,h}$ mean capacity, from the same production lot and subsequently the same grade, increase their capacity spread from $0.02\,\mathrm{A\,h}$ (1 %) to $0.1\,\mathrm{A\,h}$ (4.5 %) over 900 cycles and $0.4\,\mathrm{A\,h}$ (18 %) over 1300 cycles. The evolution exhibits significant randomness, such that a "cell that is in the lower third of the capacities after 480 cycles can end in the top ten cells after 1440 cycles ..." [19].

Whereas parallel cells even out by themselves, the usable energy of serially connected cells remains limited by the cell with the least energy. Besides supervising temperature and other cell operating parameters, a Battery Management System (BMS) hence regularly performs balancing to improve the efficiency of the pack.

A simple, but inefficient option for balancing is passive balancing where excess energy is dissipated using switchable resistors. In ACB, on the other hand, this energy is transferred to other cells within the pack. While such transfers can be implemented in many ways, this thesis focuses on inductor-based architectures since they are highly efficient with respect to energy dissipation as well as space requirement.

Balancing performance criteria There are several target variables that we may want to improve both during design phase and during operation. In this work, we mainly consider the

following objectives:

- *Minimize charge losses:* While ACB transfers charge between cells, the process is not completely lossless. The resistance of the components in the current path leads to energy dissipation, for instance. As balancing is first and foremost about energy efficiency, these losses should be as small as possible.

- *Balancing time:* In many charge transfer scenarios, the balancing time is constrained. Consider, for instance, an EV that should be balanced over night. The time for passive balancing mainly depends on the available cooling since dissipating charge leads to heat that must be removed from the system. In ACB, balancing time is mainly determined by the routing strategy and the transfer current. Adjusting the current, however, leads to a trade-off between speed and efficiency. Performing transfers in parallel is hence highly beneficial; if the ensuing speed increase is not required, lower currents can be used for more efficiency. In any case, this link between time and efficiency also implies that considering efficiency in isolation is not meaningful.

Furthermore, there are secondary objectives which, depending on the usage scenario of the battery pack, may also become relevant:

- *Raise charge level of the pack early:* The discharge of a battery pack must stop once the first cell reaches its limit. The energy level of the pack is thus determined by the weakest cell. While only a fully balanced pack leads to the maximum usable charge, there are several reasons to raise the weakest cells and thus the overall charge level early. Given that complete balancing at efficient transfer rates often requires significant time, interruptions by the user have to be expected. Consider, for instance, an EV that is parked in a shopping mall for an unknown period of time. If balancing is interrupted, the pack should already be in the best possible state. Raising the pack level quickly at the expense of some overall efficiency may thus be beneficial.

- *Minimize stress on cells:* The goal of cell balancing is to equalize the charge of cells in the battery pack to increase the usable energy. Depending on how the charge transfers are performed they may contribute differently to the wear and lifetime of the cells. Frequent transfers, high currents, or increased depth of discharge usually deteriorate a cell's health. Proper balancing may also lead to reduced depth of discharge in the weak cells, however, and potentially prolong the pack lifetime.

These criteria vary in importance depending on situation and user preferences. That said, this thesis focuses on improving the overall efficiency under a time constraint. The other goals are mostly evaluated as side effects of the strategies that achieve this purpose.

Design flow There are many challenges in the ACB area. As described in the design flow from Section 1.2, circuits must be designed, components selected and low-level functionality ensured. Although equally important, not all aspects from this flow can be considered in this thesis. Mainly, we are concerned with the quantitative modeling and optimization of the charge transfer. To provide context for the following investigation, we now briefly go over the design flow and detail what decisions we assume to be fixed and how the steps depend on each other.

1. *Topology selection:* Although many transfer mechanisms are possible, this work only deals with inductor-based circuits. Compared to their alternatives, inductors lead to high energy efficiency with relatively low space requirements. The examples also focus on modular architectures due to the focus on distributed battery management in the author's research group. Please refer to the introduction later in this section. The developed models and methods do not require modularity, however.

2. *Circuit synthesis:* From the family of inductor-based circuits, this work considers various versions. These include a basic version that only allows transfers between neighbors, but also more complicated circuits that allow transfers to non-adjacent cells or multiple cells. The higher-level DSE challenge is beyond the scope of this work and all circuits are considered given. From the perspective of the following steps, which are the focus of this work, one specific circuit differs from the other in which links it creates between cells and in the aggregated resistances there. This relationship is explained in Section 5.3.

3. *Component selection:* A circuit architecture determines only the layout and the number of Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs), inductors and possibly diodes. The final performance of the ACB circuit heavily depends on the parameters of these components. Ideally, one would want small components with vanishing resistance and high inductance. Clearly, there is a trade-off between these values involved, however, that we treat in Chapter 6.

4. *Control strategy:* Once the circuit is printed and the components are soldered, further decisions must be taken online during the balancing process. The most important questions are charge routing and transfer current. Heuristics and optimization approaches for these questions are discussed in Chapter 6.

This chapter builds quantitative models to evaluate an ACB circuit and thus lays the foundation for Chapter 6.

Modeling & simulation challenges To evaluate and subsequently improve the typical performance criteria, accurate models for computer simulation are critical. These are challenging in ACB for several reasons.

- *Time scale disparity:* The time periods for the switching phases, described in Section 5.2 (ϕ_t and ϕ_r in Fig. 5.4), are typically in the microsecond range. A balancing operation, on the other hand, may last for several hours. This means that a simulation may have to analyze domains that differ by a factor of $10\,\mathrm{h}/50\,\mu\mathrm{s} = 7.2\mathrm{e}8$.

- *Non-differentiable switching transitions:* By itself, the time scale disparity may not be a major issue. However, the switching behavior of the circuit leads to non-differentiable transitions between the individual phases. These are the kinks in Fig. 5.4 from Section 5.2. A general purpose simulator must hence increase its resolution there, severely limiting its speed overall.

- *Battery modeling:* While there exist a vast amount of battery models in the literature, most of them are concerned with the simulation of one cell or consider a pack of multiple cells as a single cell. However, since ACB is about transfers between cells, they must be represented individually and many models are not computationally efficient enough for this purpose.

Distributed battery management & smart cell architecture Most BMS implementations use a master-slave architecture. There, a powerful ECU forms the center of operation, receiving information from all sensors. At TUM CREATE, we have explored an alternative, fully distributed approach where the BMS is formed by *smart cells*.

A distributed approach imposes certain restrictions on the transfer circuits. Since no cell should have a special role, only modular circuit designs can be used. For this reason, *most examples from the thesis at hand fulfill this modularity requirement even though it is not a constraint for the presented methods.* Since balancing is a slow process and there are not too many cells, communication is not constrained and global knowledge of the state can be assumed. Message protocols that ensure charge transfers do not interfere with each other have been presented in [166].

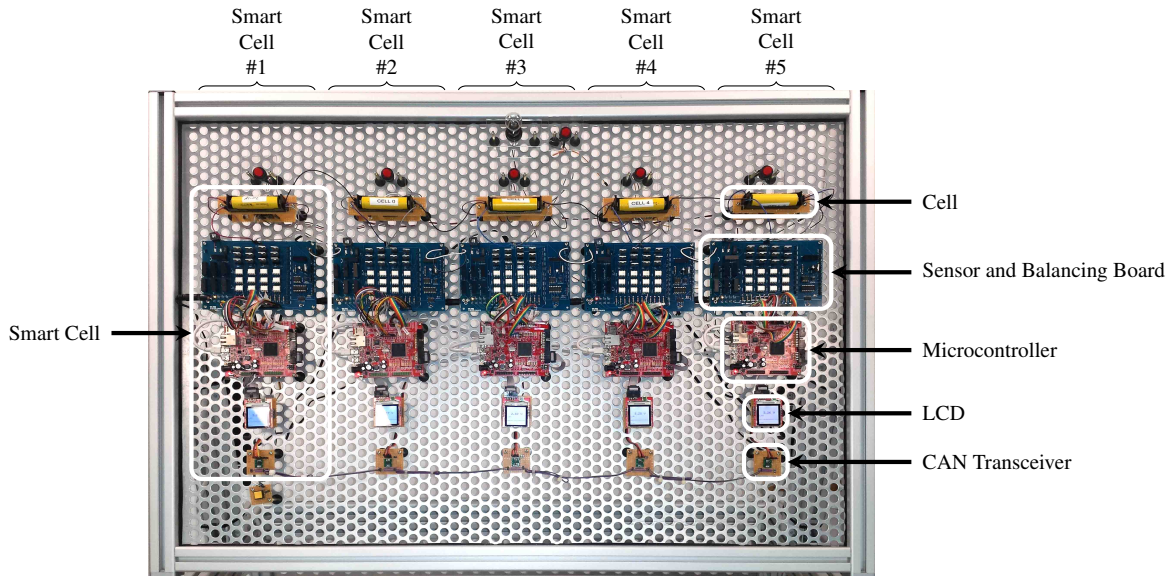


Figure 5.1: *The smart cell development platform from [167] consists of five smart cells. Each cell consists of sensor & balancing board as well as a microcontroller and CAN transceiver for communication.*

Fig. 5.1 shows a development platform with 5 cells. The communication architecture chosen for this prototypical implementation is the wired CAN bus since it is a reliable and well-established standard. The emphasis is on broadcast messages with only one smart cell transmitting to the bus at a time. Every cell thus automatically has global knowledge of the system.

The overall goal of the smart cell architecture is to come up with a single Integrated Circuit (IC) per cell, comprising the whole functionality. Integration of the computational layer and the communication layer as well as most parts for sensing and balancing is possible with

minimal footprint, low power consumption and cheap production costs. For modular active cell balancing architectures, one inductor would be required for each smart cell as in Fig. 5.2. Components for temporary energy storage like inductors cannot be efficiently integrated in a single chip, however. Packaging an inductor with the IC chip on a small Printed Circuit Board (PCB) still allows compact smart cells with negligible volume and weight of less than 50 grams. This is a small ballast compared with $2\mathrm{kg}$ per prismatic cell in state-of-the-art vehicles such as the BMW i3. The IC would also replace many conventional BMS components and the overall weight would thus remain similar in spite of scalability and integration benefits coming from the smart cell architecture.

5.2 Inductor-based charge transfer architectures

This section qualitatively presents the basic operating principles of inductor-based charge transfer. The following sections build upon these principles, adding quantitative models and more details to improve the resulting ACB performance.

Charge transfer mechanism Figures 5.2 and 5.3 show Kutkut's charge transfer circuit from [103] in a modular layout. This, arguably first, inductor-based balancing architecture has a simple topology that enables only transfers between adjacent cells. Its biggest advantage is the low number of transistors it requires. This facilitates the explanation of the general concepts behind inductor-based charge transfer. Along with the fewest transistors, the basic version from Fig. 5.2 is also the smallest in size. It hence remains relevant in practice.

The general concepts explained in this section apply to the entire family of inductor-based balancing circuits. The larger versions employ more transistors to provide additional features like transfers between non-adjacent cells or groups of cells. They always represent trade-offs, however, since the additional weight and resistance of the transistors must be justified.

Inductor-based charge transfer always occurs in several phases, each corresponding to a specific transistor configuration. Figs. 5.3 and 5.4 detail this operation. In phase ϕ_t, inductor L_2 is charged from cell c_2 by closing M_2^a. Inductor current i_L consequently rises to *peak current*

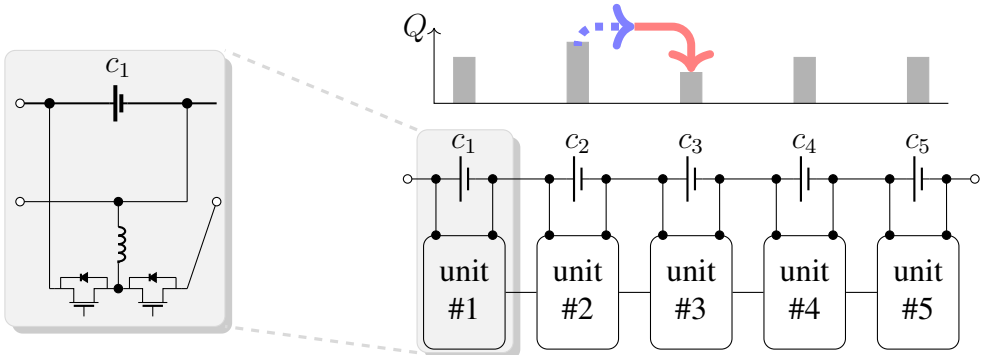


Figure 5.2: *After composing individual modules to a balancing architecture and attaching it to the main battery string, charge can be transferred between adjacent cells. Here, cell c_2 charges its inductor (▪▪▪) which subsequently discharges into c_3 (▬).*

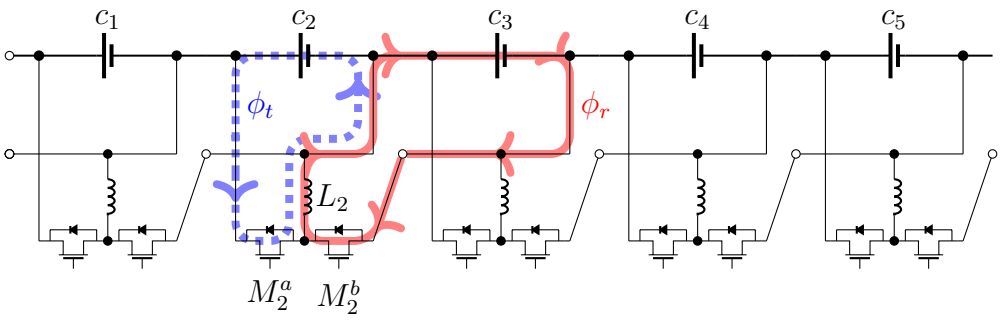


Figure 5.3: *Circuit-level perspective of the modules from Fig. 5.2: Driven by switching signals in M_2^b (see also Fig. 5.4), cell c_2 charges inductor L_2 during transmitting phase ϕ_t. The MOSFET then switches over and L_2 discharges into cell c_3 in receiving phase ϕ_r.*

I. This stores energy in the magnetic field of the inductor. Next, transistors M_2^a and M_2^b both switch over and discharging or receiving phase ϕ_r begins. The energy from the inductor now moves into another cell (c_3 in this example). Note how the direction of the inductor current i_L remains constant in Fig. 5.3. This current now decreases back to zero before M_2^b switches back to its non-conducting state. Note that charge transfer remains possible even if the voltage of c_3 is higher than that of c_2. The inductor, with its continuous current, separates the two cells and their voltages are never directly compared.

For efficient operation, it is crucial that i_L does not turn negative because that would reverse the transfer direction and extract energy from the receiving cell. In Fig. 5.4, this is achieved by M_2^b switching over precisely when $i_L = 0$. We ignore the implementation details and the timing issues for now, postponing the discussion on techniques that approximately create this ideal behavior to Section 5.5. There, we also treat model adjustments that capture the actual transfer behavior more accurately which become relevant in the case of transfers at low current or equivalently high frequency[1].

[1]The slope in Fig. 5.4 depends on cell voltage and inductance; it is hence approximately constant at runtime. We must therefore shorten T_t, the time for ϕ_t, to achieve a lower current, leading to higher frequency.

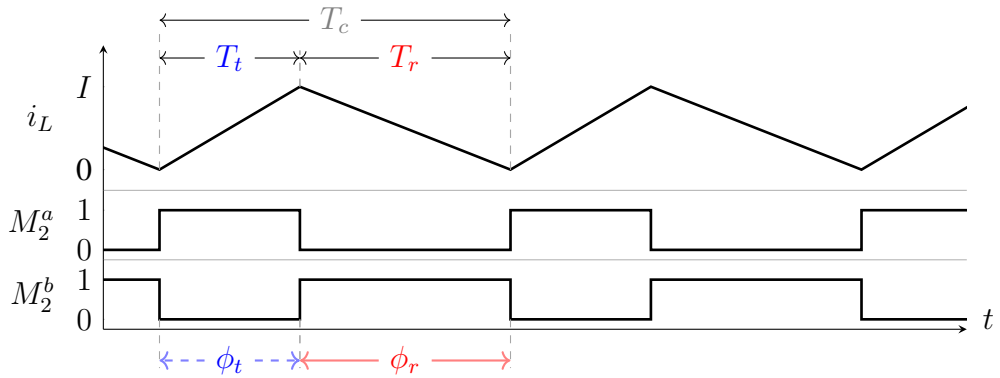


Figure 5.4: *The transfer in Fig. 5.3 is driven by switching signals in M_2^a and M_2^b. Inductor current i_L rises to peak current I during charging phase ϕ_t. Next, it goes back to zero during discharge phase ϕ_r.*

5.3 Equivalent circuit modeling

Inductor-based charge transfer occurs in several phases, as shown in Fig. 5.4. The circuits are actuated by varying the sequence timing for transmitting phase ϕ_t, and receiving phase ϕ_r. This section explains how the equivalent circuits of these phases can be considered separately to obtain an ODE model for the dynamics within each phase. These models are also referred to as *intra-phase dynamics*. Although they depend on the cell voltage, no explicit battery model is necessary for the formulations in this section. In the later analysis techniques a battery model (see Section 5.4) can then be selected as needed.

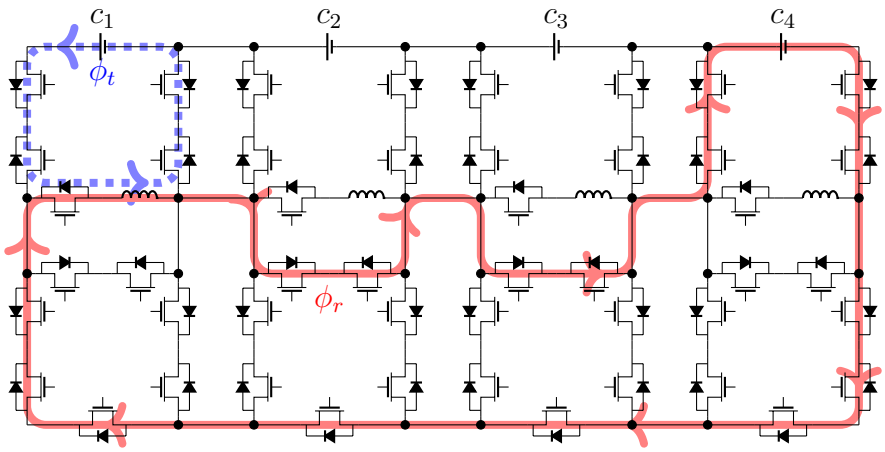


Figure 5.5: *The charge transfer circuit from [123] allows one-to-one transfers between non-adjacent cells, using several additional transistors.*

Consider the charge transfer architecture from Fig. 5.5. Compared with the basic circuit from Fig. 5.3, it contains significantly more transistors. Predictably, the switching sequence (cf. Fig. 5.4) is also more complicated, involving a total of 10 transistors that must be controlled. Please refer to [123] for details. Apart from transistor network, the components (inductor) and the modularity have remained identical. The benefit of this circuit are the non-neighbor transfers that it enables. These lead to a higher efficiency in almost all scenarios. Intuitively, this is the case because daisy chaining, the alternative for long-distance transfers in neighbor-only architectures, must temporarily charge all interim cells. However, the resistance with less transistors is lower. In practice, it thus depends on the individual application whether the additional expense and installation space for the extra components is justified.

In this section, the purpose of the more complicated architecture from Fig. 5.5 is to demonstrate that it can be analyzed with the same equivalent circuit model as the basic architecture from Fig. 5.3. Ignoring the details of the switching scheme that has been formally verified in [123], we now consider the dynamics of the individual phases ϕ_t and ϕ_r. In both architectures, the corresponding current paths contain one battery cell, one inductor and a number of MOSFETs. Since the MOSFETs do not switch during either phase, the intra-phase dynamics only need to consider their contribution to the aggregate resistance of the path. After this aggregation, the only differences between the phases are the current direction and the amount of resistance. Fig. 5.6 summarizes the involved current paths into equivalent circuits using the

aggregated transmitter resistance R_t and receiver resistance R_r. Compare this figure to both Fig. 5.3 and Fig. 5.5 to convince yourself that the circuits there represent the same circumstances (modulo cell indexes).

To quantify the dynamic behavior, formulas are required that contain the aggregated resistances for all possible transfer routes. These formulas are the interface that abstracts the full circuit schematics. Although this standard transformation forgoes several nonlinear components, it is well respected and only leads to small errors. Please refer to the measurements from Fig. 5.20 on page 109 that compare this abstraction to a standard circuit simulator.

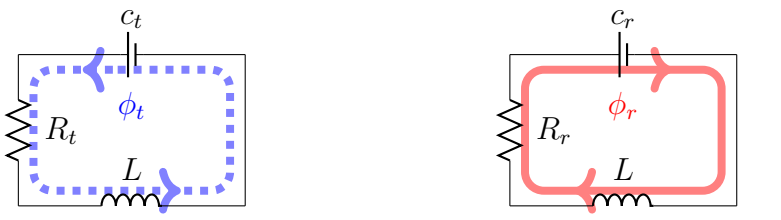
Before we can focus entirely on the dynamics of the equivalent circuits from Fig. 5.6, we must guarantee that the full circuit is free of short circuits and other design errors. Such an analysis can and should be performed with the tool presented in [123]. It formally verifies that the full circuit does not contain bugs, but it does not make quantitative statements about the performance.

Quantitative analysis is the topic of the following sections. They all build on the equivalent circuit models from Fig. 5.6. The dynamics of the transmitting and receiving cells are identical except for their parameters and current direction. The latter follows the convention in the following definition.

Definition 5.1 (Current direction)**.** *The current direction follows the convention that positive currents discharge and negative currents charge a cell. The charge differences q, being integrals of the currents, adopt the same signs. In this way, we can always add q without distinguishing cases. This is particularly important for battery models (Section 5.4) that employ additional capacitors for more accuracy.*

To simplify the equations, a short notation is often used to summarize variables of both phases. In this scheme, R refers to the aggregate resistance of both respective phases, R_t and R_r. With these definitions, the dynamics for both phase ϕ_t and ϕ_r are given as follows.

$$\frac{\mathrm{d}}{\mathrm{d}t} i(t) = \frac{1}{L}[V(t) - i(t)R] \qquad i(0) = i_0 \tag{5.1}$$



(a) Inductor charging (b) Inductor discharging

Figure 5.6: *After aggregating the resistances on the current path, the charge transfer can be described by two equivalent circuits, (a) and (b). Driven by transmitting cell c_t, the current in the inductor rises during ϕ_t until the desired peak value I is reached. The MOSFETs then switch over and the inductor discharges into the receiving cell c_r during ϕ_r.*

In this formulation, V, a short notation for V_t and V_r, refers to the respective cell voltage and must be characterized. Section 5.4 discusses the corresponding battery models and how they affect the solution of this ODE. Aggregate resistance R, short for R_t or R_r depending on which phase we are solving for, is determined by the circuit itself. For the circuit from Fig. 5.5, for instance, the transfer involving c_1 and c_4 yields the following resistances.

$$R_t = 5R_M + R_L + R_C \qquad R_r = 17R_M + R_L + R_C$$

Here, R_M, R_L, R_C refer to the resistance of the involved MOSFETs, inductors and battery cells, respectively. In general, receiver resistance R_r depends on the length of the transfer, or the distance (in cells) between the two participants. With this in mind, we can find a more general formula for this circuit.

$$R_t = 5R_M + R_L + R_C \qquad R_r(i,j) = 8R_M + |i-j| \cdot 3R_M + R_L + R_C \tag{5.2}$$

The basic, neighbor-only circuit from Figure 5.3 also has a simpler resistance function. There can be no gap between cells and all distances are hence identical $|i-j| = 1$. Concretely, its resistance values are hence given by

$$R_t = R_M + R_L + R_C \qquad R_r = R_M + R_L + R_C. \tag{5.3}$$

This is strictly less than the values from (5.2). Whether it is dramatically less depends on how large the transistor resistance R_M is in relation to R_L, R_C, the inductor and cell resistances.

More complicated circuits may implement transfers in non-symmetrical fashion to reduce the transistor count. This cannot be described by only including the cell distance $|i-j|$. Other architectures are not fully modular and come instead with two types of modules that they use in alternating fashion. A transfer $c_2 \to c_3$ is then identical to a transfer $c_4 \to c_5$ whereas a transfer $c_3 \to c_4$ would face different resistances. The equivalent circuit model from this section and the methodologies this thesis builds on top of it are valid in all these situations. The only requirement is that there is a time-invariant function

$$R : \{1,..,N\} \times \{1,..,N\} \to \mathbb{R}^2 \quad \text{with} \quad R(i,j) = \begin{pmatrix} R_t & R_r \end{pmatrix}$$

that returns the resistances for a transfer $c_i \to c_j$. Evaluating this function should be cheap to avoid interfering with other computation tasks. In practice, this is not an issue; most functions of this kind consist only of a few multiplications, additions, and absolute value evaluations.

The following definition of links, describing the connection of two cells, formalizes the interaction of high-level algorithms with the details of the corresponding equivalent circuits. In particular, the parameters of the link and the current state of the participating cells is sufficient to calculate the transfer dynamics.

Definition 5.2 (Link)**.** *A link l is identified by the transmitting cell $t(l)$ and the receiving cell $r(l)$ it connects. As parameters, a link stores the resistances R_t, R_r and the inductance L.*

We refer to the set of all links as $\mathcal{L}$. For convenience, we additionally define the following sets of links with respect to a single cell c.

$$\mathit{IN}(c) = \{l \in \mathcal{L} : r(l) = c\} \qquad \mathit{OUT}(c) = \{l \in \mathcal{L} : t(l) = c\} \tag{5.4}$$

With the link parameters R_t, R_r, L provided by the selected circuit, we are now almost ready to solve the intra-phase dynamics (5.1). The last ingredient we need is a battery model and the corresponding selection process is discussed in Section 5.4.

5.4 Electrical battery models

ACB is first and foremost concerned with Li-Ion battery cells. Their power and energy density dominate other cell chemistries in the market and make them the clear choice for many applications. The chemical reactions inside these cells lead to complicated dynamics, however, that are virtually impossible to model in their entirety. For this reason, various cell models have appeared in the literature, each catering to special use cases or circumstances. To analyze charge transfer dynamics with ODE (5.1), many of these models are not suitable because they are too complex. In most applications a cell model represents an entire pack as a single cell which justifies more computational effort for increased accuracy. This perspective cannot be taken in ACB because it is concerned with the differences of the cells. Additionally, ACB deals with high-frequency behavior and we aim to reformulate the dynamics. We hence require models that are both computationally efficient and provide insight on the equation level. This requirement rules out most cell model classes, like the chemical reaction models, or those based on neural networks. Currently, only electrical battery models are a good option for ACB since they integrate directly into the equivalent circuits from Fig. 5.6 and thus bring both equation-level analysis and fast computation.

Resistor-Capacitor (RC) battery models The most common electrical battery models are Resistor-Capacitor (RC) models. They consist of a voltage source and several RC stages in series. Prominent examples achieve small errors on the order of $0.5\,\%$ in charge content and $30\,\mathrm{mV}$ in terminal voltage under a wide range of input currents. Simpler approaches like Peukert's law, Thevenin model, or impedance circuits typically lead to errors that are at least 10 times larger [45].

Fig. 5.7 shows a cell model with a voltage source and two RC stages integrated into the equivalent circuits from Fig. 5.6. In these electrical battery models, two effects are modeled: transient voltages and main cell evolution. The RC stages, such as $C_{t,1}$ with $R_{t,1}$, model transient, often parasitic, effects. They behave according to the following ODE.

$$\frac{\mathrm{d}}{\mathrm{d}t}Q_j = -i - \frac{V_j(Q_j)}{R_j} \quad j = 1, 2 \tag{5.5}$$

When a current flows through a RC stage, (5.5) leads to a charge Q_j on C_j and consequently a voltage V_j, given by

$$V_j(Q_j) = Q_j/C_j \quad , j = 1, 2. \tag{5.6}$$

This voltage is typically parasitic and renders the main operation less efficient. During charging ($i < 0$), for instance, the overall voltage of the cell is increased by $V_j > 0$, and the charger must operate at a higher power level. When $i_t = 0$ or $i_r = 0$, the respective cell is *relaxing*. In this case, we must still update the resistor-capacitor stages using (5.5).

The main evolution of the cell is modeled by the upper circuit with the controlled current source and by the controlled voltage source V_0. The latter is also referred to as the Open Circuit Voltage (OCV). i_t directly updates the cell charge in $C_{t,0}$, and i_r analogously updates $C_{r,0}$. The resulting charge in C_0, short for $C_{t,0}$ and $C_{r,0}$, is called Q_0. It is often expressed in relation to C_0 rather than as absolute value. The so-created State of Charge (SoC) is explained in the following definition.

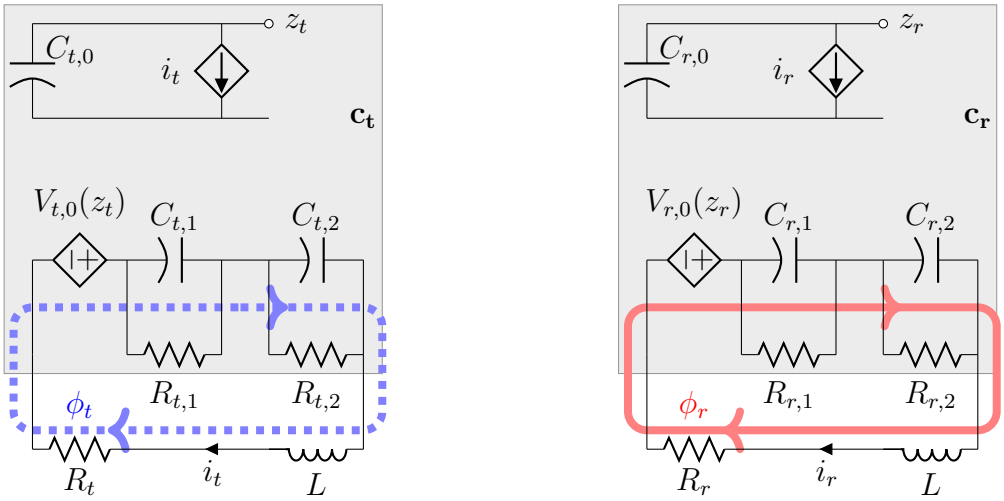


Figure 5.7: *After inserting a standard battery model into the equivalent circuit from Fig. 5.6, the transfer dynamics can still be described with two phases. There are now 8 states, however: one current and three capacitor charges per cell.*

Definition 5.3 (SoC)**.** *A cell's charge Q_0 is often equivalently expressed in percentage as State of Charge (SoC) $z = Q_0/C_0$. Absolute changes in SoC are denominated in percentage points* (pp) *or basis points* (bp)*.*

$V_0(Q_0)$, the OCV, describes the relation between a cell's charge and its voltage, without external influences. The OCV is obtained from measurements and typically expressed as piece-wise linear function or via another form of curve-fitting. A later paragraph from this section describes this in more detail.

How many RC stages? Since the RC stages discharge themselves, the maximum voltage they can reach depends on the input current. As the current is divided between capacitor and resistor such that their voltages are always identical, the voltage of the resistor represents an upper bound that the voltage of the RC stage approaches. For a constant current i_t, it holds

$$|V_j(t)| \leq \lim_{t\to\infty} |V_j(t)| = |i_t|\, R_j \tag{5.7}$$

This can be seen from (5.5) where the right-hand side vanishes for $V_j = -iR_j$. When deciding how many RC stages should be used in a model, this upper bound becomes useful.

Since currents in ACB tend to be low, operating with few or even no RC stages is not an issue in many cases. Typical peak currents of $\frac{1}{4}C$ lead to less than $10\,\mathrm{mV}$ in RC contributions[2]. The RC voltages may reach $50\,\mathrm{mV}$ only with high C rates (see Fig. 5.22 in Section 5.8.3). As these differences are dwarfed by the minimum cell voltage and even by changes in OCV, RC stages are rarely included in ACB optimization and often omitted for simulation as well.

[2] A peak current of $\frac{1}{4}C$ corresponds to $\frac{1}{16}C$ on average (factor $\frac{1}{2}$ for idling during the opposite phase and another factor $\frac{1}{2}$ for the triangle shape of the current). The voltage estimate is for the cells characterized in [45].

Piecewise linear charge-voltage mapping In electrical battery models, the Open Circuit Voltage (OCV) describes the major evolution of the cell. It is the terminal voltage of a cell without load and after transient effects have worn off. Correspondingly, it can be seen in Fig. 5.7 that V_0 becomes the terminal voltage of the cell once the voltages in C_1, C_2 have decayed. The OCV is a mapping $Q \to V(Q)$ that must be established from measurement before operation begins. This mapping is crucial because the SoC, or equivalently the charge of the cell cannot be measured directly. These values must hence be estimated from the measured terminal voltage, the recent current measurements, and knowledge about the OCV at runtime.

To see how an OCV mapping can be devised, consider the measurements in Figs. 5.8 and 5.9. They show voltage evolutions over the corresponding SoC during a slow complete discharge. Both measurements were performed at approximately $0.1C$. The C-rate defines the discharge current in relation to the capacity of a cell. $1C$ refers to a current that discharges the cell within one hour; a current of $0.1C$ correspondingly requires 10 hours for a full discharge. The difference between Fig. 5.8 and Fig. 5.9 is the battery cell used for the measurement. The APR18650M1 cell from A123SYSTEMS [1] has a Lithium Iron Phosphate ($LiFePO_4$) chemistry. This chemistry leads to very high power and abuse tolerance. It also possesses a very flat voltage profile, making SoC estimation difficult. Its capacity of $1.1\,\mathrm{A\,h}$ is significantly less than the $2.5\,\mathrm{A\,h}$ of the newer INR18650-25R cell from SAMSUNG [156]. The latter uses a Lithium Nickel Cobalt Aluminum Oxide ($LiNiCoAlO_2$) chemistry, also referred to as NCA. This composition offers the highest specific energy while retaining acceptable specific power.

Both cells have the same 18650 form factor, indicating a cylindrical shape with a diameter of $18\,\mathrm{mm}$ and a length of $65\,\mathrm{mm}$. With their weight below $50\,\mathrm{g}$, an EV can employ thousands of them to achieve the desired capacity. Tesla has chosen such a massively parallel setup of ($LiNiCoAlO_2$) cells over larger battery cells, for instance. In a laboratory environment, the most important thing is that these cells are both cheaper and far safer to use than their larger counterparts.

Under low currents, both cells exhibit an approximately linear decrease in voltage with decreasing SoC. For the SAMSUNG cells there are two kinks around 15% and 5%. As Fig. 5.9

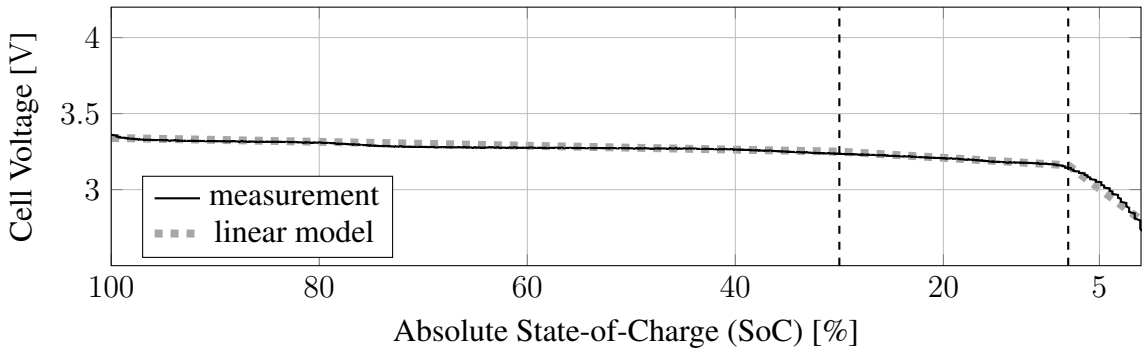


Figure 5.8: A123SYSTEMS APR18650M1 *($LiFePO_4$ chemistry) – 10h discharge measurement (corresponding to a C-rate of 0.1) and piecewise linear charge-voltage mapping*

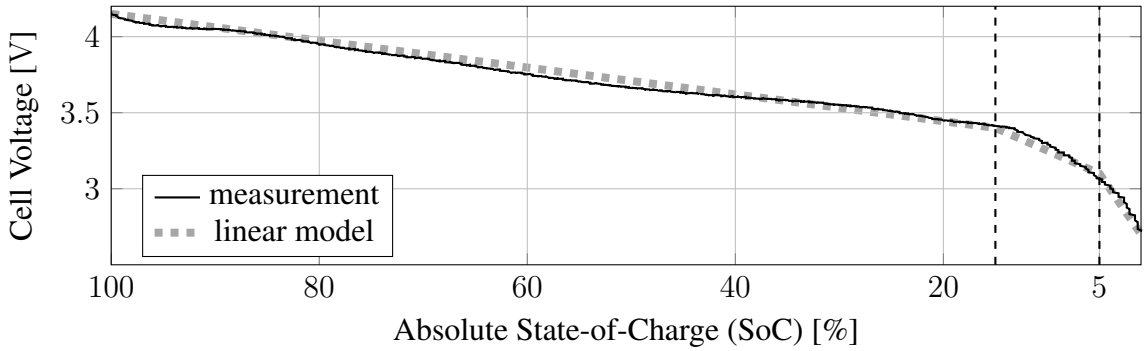


Figure 5.9: SAMSUNG INR18650-25R *(LiNiCoAlO$_2$ chemistry) – 10h discharge measurement (corresponding to a C-rate of 0.1) and piecewise linear charge-voltage mapping*

shows, this behavior can be captured well by a small piecewise linear model.

$$V(Q) = \begin{cases} V_{\zeta,0} + \zeta_0(Q - 0) & \text{if } Q < Q_1 \\ V_{\zeta,1} + \zeta_1(Q - Q_1) & \text{if } Q_1 \leq Q < Q_2 \\ V_{\zeta,2} + \zeta_2(Q - Q_2) & \text{if } Q_2 \leq Q < Q_{\text{max}} \end{cases} \tag{5.8}$$

For the SAMSUNG measurement, we have

$$\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} 0.05Q_{\text{max}} & 0.15Q_{\text{max}} \end{bmatrix} \qquad \begin{bmatrix} V_{\zeta,0} & V_{\zeta,1} & V_{\zeta,1} \end{bmatrix} = \begin{bmatrix} 2.5 & 3.1 & 3.4 \end{bmatrix}. \tag{5.9}$$

Given $V_{\zeta,3} = V_{\text{max}} = 4.15$ and $Q_{\text{max}} = 2.5\,\text{A h}$, ζ_i are then calculated such that the curve is continuous. The A123SYSTEMS measurement (Fig. 5.8) yields

$$\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} 0.08Q_{\text{max}} & 0.3Q_{\text{max}} \end{bmatrix} \qquad \begin{bmatrix} V_{\zeta,0} & V_{\zeta,1} & V_{\zeta,1} \end{bmatrix} = \begin{bmatrix} 2.7 & 3.16 & 3.25 \end{bmatrix}. \tag{5.10}$$

Further parameters in that piecewise linear model are $V_{\zeta,3} = V_{\text{max}} = 3.34$ and $Q_{\text{max}} = 1.1\,\text{A h}$.

While this work uses a piecewise linear approach for the OCV, other techniques like curve-fitting with smooth functions work equally well for simulation. For optimization in Chapter 6 and also for one long-term simulation approach in Section 5.8.4, locally linear behavior is required for the reformulations.

Modeling a limited SoC region If the OCV model only needs to model a certain SoC region, it can become even simpler which is particularly helpful for optimization tasks. In ACB, different such regional restrictions are justifiable. The first option is limiting the model to the commonly used SoC range between 20% to 80% of absolute SoC. Cells are under greater stress and age faster if they operate in the highest or lowest 20% of the absolute SoC range. These areas are thus rarely used because life time requirements take priority and do not need to be modeled in most scenarios. Another option is adjusting the model according to the current SoC distribution. As battery cells in a pack typically do not deviate too far from each other and move towards each other during balancing, it may be preferable to model only the region $[\min_i(z_i), \max_i(z_i)]$ between the minimum and maximum SoC of the pack.

Reevaluating Eq. (5.9) and Eq. (5.10) with a restriction to the most common SoC region (20% to 80%), we find that both of them can be treated as linear models. In the SAMSUNG case, this is evident from the parameters in (5.9) themselves. In the A123 case, Fig. 5.8 helps visualize that the segment transition at $z = 30\%$ is not a crucial one. If we further restrict the OCV model according to the current SoC distribution, on the other hand, the typical SoC deviation of 2-5% leads to linear models being accurate for virtually all Li-Ion cells.

Restricting the support region of the OCV model in one of these ways is particularly relevant for optimization. There, a nonlinear model or even the kinks of a piecewise linear model usually render the solution significantly more expensive. For simulation methods, by contrast, the changeovers of the piecewise linear approach are not a major issue. They should hence be included for accuracy reasons.

Cell energy With battery model (5.8), the energy stored in a cell that is currently within the i-th linear segment can be calculated as follows.

$$\begin{aligned} E_{\mathrm{B}}(Q) = \int_0^Q V(q)\,\mathrm{d}q = E_{\zeta,i} + \int_{Q_i}^Q V_{\zeta,i} + \zeta_i(q - Q_i)\,\mathrm{d}q \\ = E_{\zeta,i} + \frac{1}{2}(Q - Q_i)(2V_{\zeta,i} + (Q - Q_i)\zeta_i) \end{aligned} \tag{5.11}$$

Recognizing that, from Eq. (5.8), it also holds that $Q = \frac{V - V_{\zeta,i}}{\zeta_i} + Q_i$, this further transforms to

$$E_B(V) = E_{\zeta,i} + \frac{1}{2\zeta_i}(V^2 - V_{\zeta,i}^2). \tag{5.12}$$

5.5 ACB actuation interfaces

Section 5.2 introduces the basic transfer mechanism in inductor-based circuits for ACB. This description presents two phases ϕ_t and ϕ_r that immediately follow each other (Fig. 5.4) and correspond to equivalent circuits (Fig. 5.6). The circuit is actuated by varying the times T_t, T_r, and $T_c \geq T_t + T_r$. Not all combinations lead to desirable results, however. In the following, we hence discuss three actuation interfaces that build on this basic input method. The first simply selects timing parameters and then leaves them constant for a certain period of time Δt. This requires a a break period where the inductor discharges over a diode to ensure a safety margin against undesired discharge of the receiving cell. Alternatively, the inductor can be entirely discharged over a diode which represents an inefficient, yet important special case for experimentation that can be set up quickly. The second interface specifies the peak current and then continuously adjusts the timing parameters at runtime. Although more complicated to implement, this approach approximately achieves the ideal waveform from Fig. 5.4. The third builds on the second interface. Instead of specifying the current, it asks for transfer amount and time limit to calculate the best current online.

5.5.1 Fixed timing actuation

The most basic interface requires no equations to operate in practice. The user directly adjusts the timing parameters T_t, T_r and $T_c \geq T_t + T_r$. An additional parameter Δt determines for how long the system should perform work with these settings before returning for further instructions. Since the time to discharge the inductor, is not precisely known a priori and changes over time, a fixed timing must include a break period where the inductor discharges over a diode. This diode prevents undesired discharge of the receiving cell. In most cases, a diode included with a MOSFET may be used for this technique and no additional components are needed.

Fig. 5.10 provides a detailed view on the various phases that become necessary with fixed timing. Compared to Fig. 5.4, it contains the additional break phase ϕ_b. To leave a safety margin, T_r is selected shorter than necessary to deplete the inductor. Phase ϕ_r thus ends with a small remaining current I_b. Inductor current i_L then proceeds to decrease with a steeper slope because of the additional voltage drop at the diode V_d. Once the current reaches zero, the diode becomes non-conducting and an undesired discharge of receiving cell c_r is prevented. The corresponding equivalent circuits from Fig. 5.6 must be updated to include ϕ_b. As shown in Fig. 5.11, ϕ_b can be treated like ϕ_r after the addition of the diode drop voltage V_d.

The parameters for fixed-timing actuation as in Fig. 5.10 have been summarized in the following table. In this table, derived values, like I, I_b, and T_b, indicate their main runtime dependency in parentheses. i_d refers to the desired current at the end of a phase.

In total, we are thus dealing with three phases under a fixed timing actuation scheme. Adding a third phase complicates many subsequent calculations significantly, however. Even when a diode is used to prevent undesired discharge in this manner, one should thus always consider whether the required safety margin is small enough to ignore ϕ_b in simulation and optimization.

Another fixed timing approach with only two phases is popular in lab environments. By

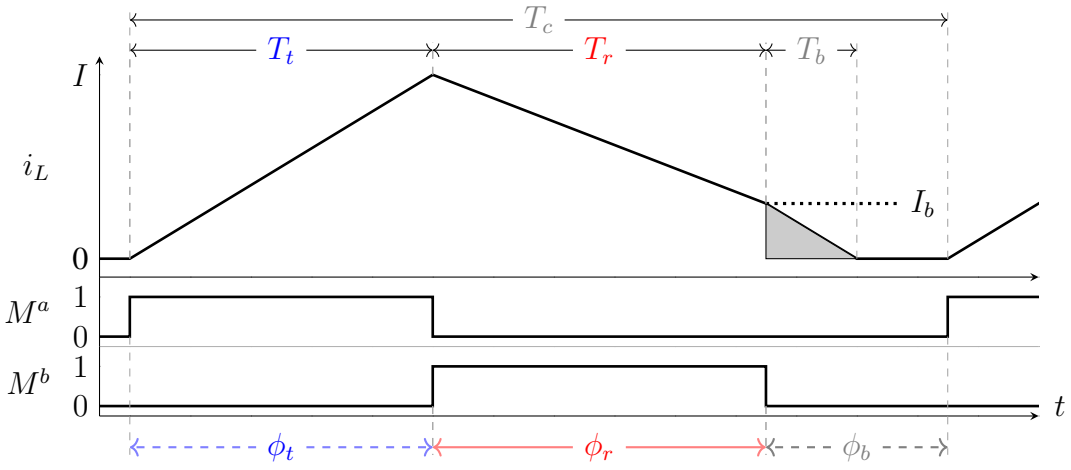


Figure 5.10: *With actuation timing fixed over prolonged periods, a break period ϕ_b is necessary where a diode prevents a reversal of the current (see also the ideal current shape from Fig. 5.4). The remaining current I_b and the corresponding charge q_b (gray area) may become significant enough to be included in simulation models.*

(a) Inductor charging (b) Inductor discharging

Figure 5.11: *Fixed-timing actuation requires a safety margin and hence transfers over a diode for a non-negligible period of time. This creates a third phase that must be taken into account by adding a diode to the equivalent circuits from Fig. 5.6.*

operating the entire receiving phase ϕ_r over a diode, the additional phase transition is no longer necessary. Since the diode greatly reduces efficiency in this way, the approach is rarely used in real applications. Its advantage is that it only needs one control signal (M^a) and very few components. For experiments, it is hence often the method of choice because it can be implemented so quickly.

After the implementation details, the following definition summarizes how a user or a higher-level algorithm may interact with the ACB platform according to the fixed timing interface.

Definition 5.4 (T-Movement)**.** *A time-based movement*

$$m = \begin{bmatrix} t & r & T_r & T_t & T_c & \Delta t \end{bmatrix}$$

specifies which link – identified by transmitter t and receiver r – to operate with a given timing for a certain duration Δt. The timing parameters specify the length of inductor charging T_t, the length of discharging T_r and the overall cycle time cycle time T_c (see also Fig. 5.10). As the exact break time T_b is not known a priori, T_c should be chosen with a safety margin.

Fig. 5.12 illustrates the feed-forward process to operate such a time-based movement m. With fixed timing parameters, transfer dynamics (5.1) with a battery model can be actuated directly. The unknown time T_b (see Fig. 5.10) for the diode discharge presents a challenge only for simulation. This is addressed individually by the approaches from Section 5.8.

Table 5.1: *Phase-dependent parameters for charge transfer dynamics* (5.1) *in the T-interface, including diode voltage V_d and break current $I_b = i(T_r)$ for ϕ_b (see also Fig. 5.10)*

Phase	V	T	i_0	R	i_d
ϕ_t	V_t	T_t	0	R_t	$I(T_t)$
ϕ_r	V_r	T_r	$-I$	R_r	$I_b(T_r)$
ϕ_b	$V_r + V_d$	$T_b(I_b)$	$-I_b$	R_r	0

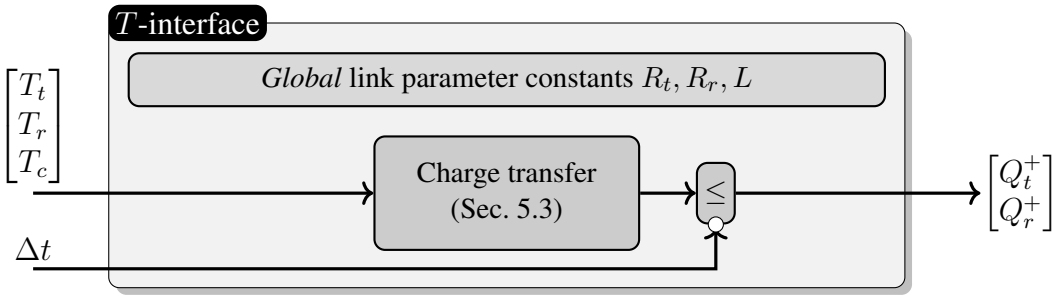


Figure 5.12: *The timing-based T-interface expects user-specified timing parameters (see Def. 5.4). The system is then actuated with these fixed settings in a feed-forward fashion over a time frame Δt before reaching a new state $\begin{bmatrix} Q_t^+ & Q_r^+ \end{bmatrix}$.*

It is advisable to use a timing formula like (5.20) from Section 5.7 when determining the input parameters. It is ultimately up to the user, however, to ensure the selected timing does not lead to unsuitable currents.

5.5.2 Current interface

Instead of asking the user to supply timing parameters that lead to a certain current behavior, the platform can also calculate the corresponding low-level settings by itself. Although it would be desirable to specify the average current from a modeling perspective, there are several issues with this choice. Dissipative losses make it challenging to calculate the timing parameters that correspond to a certain average current. More importantly, different voltages at the transmitting and the receiving cell lead to different average currents as the inductor charges and discharges faster for higher voltages. This thesis hence prefers to specify peak current I which allows a lossless transformation to timing parameters and which is shared by both transmitting and receiving side.

There are several techniques to achieve a specified peak current at runtime. Using a current sensor, an analog comparator can automatically switch the involved transistors at the right moment. Digital feedback control can similarly adjust the timing parameters over time, preferably by averaging several measurements. As current sensors are either expensive or intrusive, they are often not an option, however. In this case, the timing can be calculated with reasonable accuracy based on standard voltage measurements and a mathematical model like (5.20) from Section 5.7.

It is understood that all actuation approaches still operate with the three phases shown in Fig 5.10. The control techniques achieve an extremely short duration for break phase ϕ_b, however. This is not only favorable from an efficiency point of view, but also makes the modeling of ϕ_b during the design stage unnecessary. Overall, the ideal current behavior from Fig. 5.4 is hence approximately achieved. Table 5.2 summarizes the parameters for fixed-current actuation. In contrast to Table 5.1, the timing now depends on peak current I.

Fig. 5.13 shows the current-based, or I-interface built on that formula. It describes on-line actuation and simulation for a single link. In this setting each link manages timing questions on its own. For this purpose, it receives the evolving voltage data to adjust the timing at runtime,

Table 5.2: *Phase-dependent parameters for charge transfer dynamics* (5.1) *following the I-interface (see also the current diagram in Fig. 5.4 on page 85)*

Phase	V	T	i_0	R	i_d
ϕ_t	V_t	$T_t(I)$	0	R_t	I
ϕ_r	V_r	$T_r(I)$	$-I$	R_r	0

using a timing formula like (5.20) from Section 5.7. Even more accurate results are possible with current measurements, as described at the beginning of this section.

A user or higher-level algorithm now only needs to supply peak current I, a single and far more tangible parameter than the previous timing details. The following movement definition summarizes the interaction with the ACB platform under the fixed current paradigm.

Definition 5.5 (I-Movement)**.** *A current-based movement*

$$m = \begin{bmatrix} t & r & I & \Delta t \end{bmatrix}$$

specifies which link – identified by transmitter t and receiver r – to operate with a given peak current I for a certain time Δt.

Besides minimizing diode involvement and eliminating inefficient break period, the I-interface has another mathematical advantage. The two phases ϕ_t and ϕ_r can be analyzed in an entirely separated fashion. Since the peak current in the T-interface evolves over time as the sender cell voltage V_t decreases, the calculation of c_r must take this evolution into account. This is unnecessary in the I-interface where the peak current is maintained at a constant level. This does not mean that the involved cells, c_t and c_r, can also be separately simulated, however. As the phase lengths T_t and T_r adjust to the current specification and change over time (see also Table 5.2), knowledge about the other cell is required to analyze the evolution of a cell in the time domain. This difficult timing behavior is the disadvantage of the I-interface. The simulation approach from Section 5.8.3, for instance, tracks time separately for this reason.

5.5.3 Energy block interface with platform-determined current

The I-movement where the transfer rate instead of low-level timing behavior is specified provides a more straightforward user experience. These advantages notwithstanding, selecting the best current by hand may still prove complicated.

Searching for an optimal current appears fruitful, however, as Fig. 5.14 motivates. There are essentially two kinds of losses that need to be taken into account for this purpose. On the one hand, there is dissipation in the resistances contributed by all the involved components. These losses behave akin to a resistor and grow quadratically in the current. In other words, they are approximately proportional to "I^2R". On the other hand, operating the transistors leads to switching losses. The corresponding models are summarized in Section 5.6. As the textbook model has a constant component that is lost during each switching cycle and the frequency increases with lower current, these losses grow proportionally to "$1/I$". It is intuitively clear that the overall losses hence have a minimum value that is achieved by some optimal current.

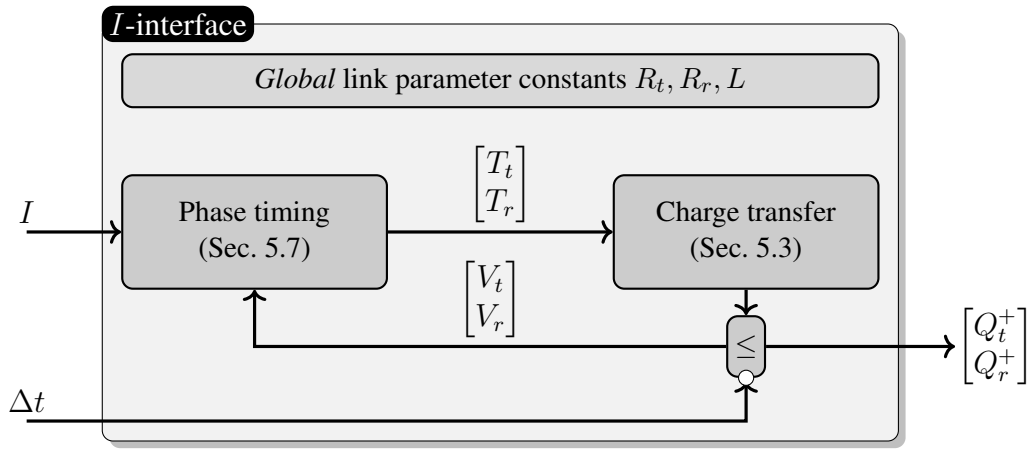


Figure 5.13: *In the current-based I-interface, the user specifies peak current I and transmission time Δt (see Def. 5.5). The system then derives the timing parameters automatically, ideally adjusting after each cycle as the voltages evolve. Once Δt has elapsed, the new state $\begin{bmatrix} Q_t^+ & Q_r^+ \end{bmatrix}$ is reached.*

Minimizing losses per time may not lead to the result that a user actually wants, however. After all, a smaller current will lead to a longer transfer time and higher total losses may subsequently be accumulated over this longer period. This motivates looking at the losses per transferred amount to minimize the energy expenditure for equalizing two given cell levels irrespective of the corresponding time requirement. Informally speaking, this roughly entails a division by I of the loss terms in Fig. 5.14. The involved functions then behave similarly to "$1/I^2$" and "RI", respectively.

Whether optimizing for losses per time or per transferred amount, the optimal current depends on many variables. Some, like the resistances, depend on which cells are involved, others, like the sender and receiver voltage V_t, V_r change at runtime. It is thus sensible to have the platform calculate optimal currents online. While we postpone a detailed discussion on the computational options to Section 6.3, the following paragraphs explain how to interact with a platform that has these capabilities, using the ΔE-interface.

Fig. 5.15 illustrates how the ΔE-interface builds on top of the I-interface. The user speci-

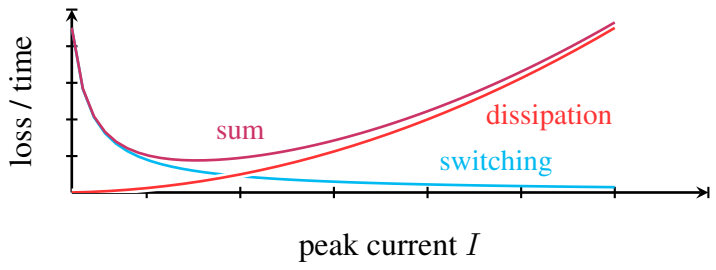


Figure 5.14: *Following a shape (or convexity) argument, there must be an optimal current minimizing the sum of transfer (dissipative) and switching losses. Lower currents lead to higher switching frequency and consequently losses while dissipation rises with increasing currents. There must hence be a current where their sum is minimized.*

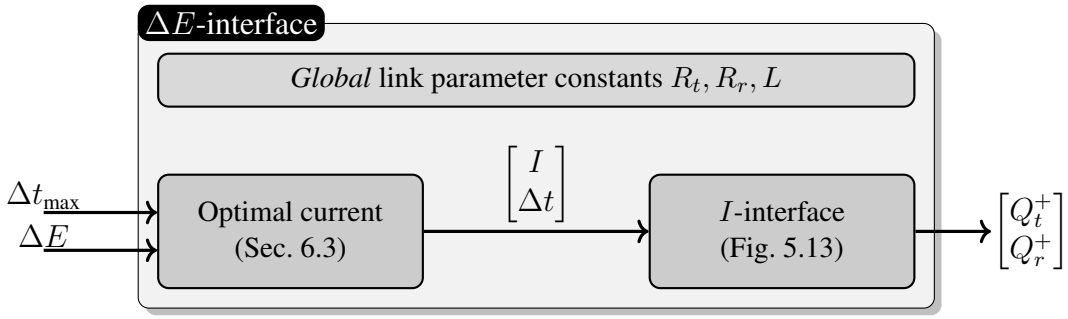


Figure 5.15: *Energy-based actuation interface: Given an amount of energy ΔE to be transferred over a certain link under a time constraint Δt_{max}, the platform calculates an optimal current I_{opt} with a corresponding actuation time Δt for this transmission. The low-level timing details for the chosen current level are subsequently determined by the I-interface. After simulating the link for a duration of Δt, this yields the new state $\begin{bmatrix} Q_t^+ & Q_r^+ \end{bmatrix}$.*

fies how much energy should be moved over a certain link. Additionally, he defines a bound on transmission time Δt_{max} that he deems acceptable instead of a fixed transmission time Δt which implicitly sets the amount to transfer in the current-based movement (Definition 5.5). Once the platform has determined the most suitable current, it operates by forwarding this result to the I-interface. In this process, Δt_{max} is only a constraint and the system will finish faster than specified, leading to $\Delta t < \Delta t_{\text{max}}$, if it is advantageous from an energy perspective. The following movement definition formalizes the interaction with an ACB platform under this paradigm.

Definition 5.6 (E-movement)**.** *An energy-based movement is given by*

$$m = \begin{bmatrix} t & r & \Delta E & \Delta t_{max} \end{bmatrix}.$$

It specifies a certain amount of energy ΔE to be transferred during a maximum time frame Δt_{max} from transmitter t to receiver r.

5.6 Freewheeling phases & switching losses

In addition to the diode involvement described in Section 5.5.2, there are several other effects that may become relevant beyond the dynamics described by the equivalent circuit model from Section 5.3. Here, we discuss the influence of the transistor network on the transfer behavior although it only becomes significant for low currents. Diodes and particularly transistors have highly complex behavior that is almost impossible to model in its entirety. For ACB simulation, we hence focus only on the most relevant effects: freewheeling and switching losses. Freewheeling phases become necessary because transistors cannot switch instantaneously and conducting phases that directly follow one another would thus lead to short circuits. In addition to being non-instantaneous, the transitions of a transistor are also not lossless. The following passages discuss how freewheeling phases are implemented and how the accompanying switching losses can be quantitatively taken into account during simulation and optimization.

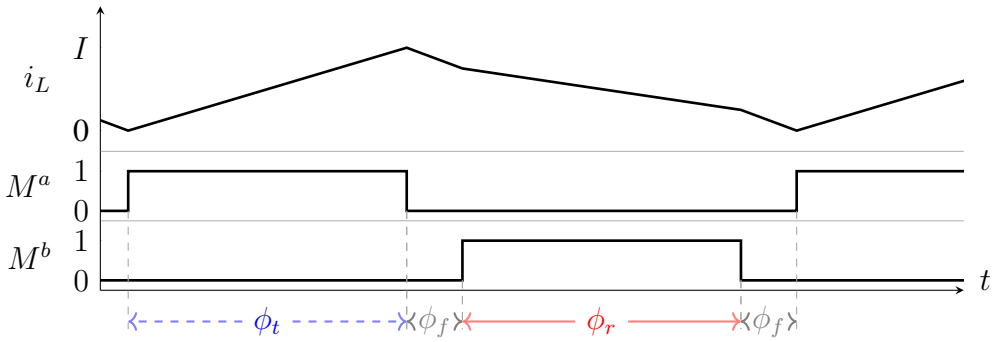


Figure 5.16: *Non-overlapping switching signals for ACB transfers contain short freewheeling phases where diodes are used to avoid short circuits.*

Fig. 5.16 provides a more detailed view on the various phases during the inductor current evolution than Fig. 5.4. In phase ϕ_t, the inductor is charged from cell c_t by closing M^a. It reaches *peak current* I before M^a switches over and phase ϕ_f begins. In ϕ_f, the inductor briefly discharges into cell c_r over a freewheeling diode. M^b then activates in non-overlapping fashion and the main discharging occurs in phase ϕ_r, without involving the diode and thus at higher efficiency. Finally, M^b is deactivated again and another freewheeling phase is necessary to transition back to ϕ_t.

Freewheeling is necessary because transistor switching is not instantaneous. A $7.8\,\mathrm{A}$-rated power MOSFET from ON Semiconductor [141], for instance, requires a total time of $14\,\mathrm{ns}$ to turn on. This summarizes turn-on delay and rise time from the data sheet. We denote this switching delay τ_u because it refers to the "up" movement. Similarly, there is a switching delay for the "down" part τ_d that consists of turn-off delay and fall time. For the aforementioned transistor, τ_d has a value of $24\,\mathrm{ns}$. A switching scheme must wait at least τ_d after switching off M^a before switching on M^b. Otherwise, a short circuit occurs because both transistors conduct simultaneously. M^b only switches on after a further delay τ_u, however. Freewheeling phase ϕ_f hence has a minimum length of $\tau_u + \tau_d$.

If a diode is necessary to avoid a reversal of the current, the second freewheeling phase becomes longer and effectively turns into break phase ϕ_b as in Fig. 5.10. The analysis of this case is discussed in Section 5.5.2.

As long as the freewheeling phases remain on the order of the transistor switching times, i.e., close to the minimum length, it is common to model the involved transition effects only as switching loss. Please refer to Chapter 4.3 "Switching Losses" in Erickson's power electronics textbook [63] for more detailed information on transistor switching. MOSFET switching losses in ACB consist of two main components: charging of output capacitances plus current during transition periods. The energy dissipated for charging output capacitance of a MOSFET is given by $\frac{1}{2}C_{OSS}V_{ds}^2$. In this formulation, C_{OSS} and V_{ds} are output capacitance and drain-source voltage of the transistor, respectively. In addition, the current that is drawn during τ_u – summarizing turn-on delay and rise time – and τ_d – consisting of turn-off delay and fall time – cannot be utilized. This entails losses of the form $\frac{1}{2}\tau I_{ds}V_{ds}$ where I_{ds} is the drain-source current of the transistor.

Following this textbook approach, the transistor of the transmitter and the receiver expend

an additional switching energy $E_{sw,t}$ and $E_{sw,r}$, respectively, where

$$E_{sw,t} = \frac{1}{2}\tau_d I V_t + \frac{1}{2} C_{OSS} V_t^2 \qquad E_{sw,r} = \frac{1}{2}\tau_u I V_r + \frac{1}{2} C_{OSS} V_r^2. \tag{5.13}$$

The current-related terms correspond to the end of ϕ_t and the beginning of ϕ_r where $i(t) = I$. The current terms related to the transition in ϕ_b, i.e., from the start of ϕ_t and the end of ϕ_r have been ignored in these formulations because $i(t) \approx 0$ there.

In many cases, the switching losses are more conveniently tracked in terms of charge such that the main dynamics can directly take them into account. This formulation can be achieved via division of energy by voltage $q = E_{sw,t}/V_t$.

$$q_{sw,t} = \frac{1}{2}\tau_d I + \frac{1}{2} C_{OSS} V_t \qquad q_{sw,r} = \frac{1}{2}\tau_u I + \frac{1}{2} C_{OSS} V_r \tag{5.14}$$

More recently, Xiong has shown that the $C_{OSS}V^2$ terms in (5.13) are too conservative for power MOSFETs with high currents (see [189]). It appears that part of the losses during the transition phase (the $\frac{1}{2} t I_{ds} V_{ds}$ terms) actually charge the parasitic capacitances. By adding $\frac{1}{2} C_{OSS} V_{ds}^2$ terms a calculation may hence count these losses twice. Before introducing a more complex model, the authors demonstrate that using

$$E_{sw,t} = \frac{1}{2}\tau_d I V_t \qquad E_{sw,r} = \frac{1}{2}\tau_r I V_r \tag{5.15}$$

is superior to (5.13) and sufficient for most applications in this environment. From an optimization point of view, the shorter expression is preferable because it is linear in both I and V once the other is fixed. The corresponding $q_{sw,t}$ and $q_{sw,r}$ as in (5.14) are similarly simpler. This makes many calculations easier and more efficient. In the end, it depends on peak current and MOSFET type whether (5.13) or (5.15) is more appropriate.

The freewheeling phases may also be completely ignored during the modeling and analysis stage in certain cases, even though they are crucial for operation. This is because T_t, T_r the lengths of phases ϕ_t, ϕ_r ($\approx 100\mu s$) are typically several orders of magnitude larger than the switching times τ_u, τ_d ($< 100\mathrm{ns}$). At low currents, however, the switching losses may become significant as frequency increases and leads to shorter phase times. If an optimization technique explores many currents, the underlying model should hence include switching terms in some way.

5.7 Transfer dynamics assuming constant voltage

The individual phases of an ACB charge transfer can be modeled by an equivalent circuit with aggregated resistances (Section 5.3). While the battery cell itself can be modeled by electrical battery models (Section 5.4), it is also beneficial to analyze the intra-phase dynamics (5.1) for a constant voltage. This approach is justified since the internal resistance of the cell is already modeled by its contribution to the aggregated resistance R_t or R_r and because the individual phases are in the microsecond range. This perspective tremendously simplifies the ODE solution and leads to important formulations for both actuation and simulation. We evaluate the errors from this simplification in Fig. 5.21 on page 110 alongside other results.

After modeling the cell as a constant voltage source with series resistance, the equivalent circuit from Fig. 5.6 becomes Fig. 5.17. This model should only be used for the dynamics of a single phase. The results from this section can, however, be used in conjunction with a more sophisticated battery model as iterative simulation technique that calculates the state evolution phase by phase.

Current and charge evolution With constant voltage V, both charging (ϕ_t) and discharging (ϕ_r) phase are governed by the following ODE.

$$\frac{\mathrm{d}}{\mathrm{d}t} i(t) = \frac{1}{L}[V - i(t)R] \qquad i(0) = i_0 \tag{5.16}$$

With its single differential operator, this is a first-order ODE. Note that the ODE parameters V, i_0, R need to be adjusted according to the individual phase as the system evolves. These values depend on the actuation approach as discussed in Section 5.5. They have been summarized in Table 5.1 and Table 5.2. With these parameterization questions taken care of, the variation of constants technique finds the unique solution of (5.16). It is given by

$$i(t, V, i_0, R) = \frac{V}{R} - \frac{V - i_0 R}{R} \exp\Big(\frac{-R}{L} t\Big). \tag{5.17}$$

This describes the evolution of the inductor current provided the involved MOSFETs never switch. For more long-term studies, the amount of charge that is moved during one phase is even more important. To calculate this quantity, we integrate current $i(t)$ from (5.17). This yields the evolution of the transferred charge over time.

$$q(T, V, i_0, R) = \int_0^T -i(t, V, i_0, R)\,\mathrm{d}t = -\frac{V}{R} T - \frac{L(V - i_0 R)}{R^2}\Big[\exp\big(\frac{-R}{L} T\big) - 1\Big] \tag{5.18}$$

Here, the integration constant is chosen such that $q(0) = 0$ and the sign of q follows the convention from Definition 5.1.

In addition to the charge from inductor current i in (5.18), potential RC stages of the battery model (see Section 5.4) must consider their self-discharge. Integrating that self-discharge term from (5.5), we obtain

$$q_j(T, Q_j) = -\frac{V_j(Q_j)}{R_j} T. \tag{5.19}$$

Phase timing Eq. (5.18) serves as closed-form equation for the charge evolution within a switching phase. As such, it is sufficient to drive simulations once the switch timing is given. Not all timing settings are meaningful, however. To determine timing parameters, it is helpful to know the time required to reach a certain current.

$i(t)$ from (5.17) rises monotonically and reaches a desired current i_d after a period of T_d, i.e., $i(T_d) = i_d$. With the settings of this section (constant intra-phase voltage leading to first

(a) Inductor charging (b) Inductor discharging

Figure 5.17: *As individual phases are brief, the battery cells from Fig. 5.6 can be replaced with constant voltage source plus series resistance there.*

order ODE), the inductor current $i(t)$ is invertible with respect to time t. More concretely, the time T_d can be calculated as

$$T_d(i_d, V, i_0, R) = \frac{-L}{R} \log\left(\frac{V - i_d R}{V - i_0 R}\right). \tag{5.20}$$

This equation is applicable in the fixed current interface from Section 5.5.2. In addition, it can calculate ϕ_b from Section 5.5.1 and help with a priori timing design.

If the timing parameters are adjusted continuously, timing calculation and charge transfer can be integrated into a single step for faster simulation and easier analysis. Substitute T_d from (5.20) into the equation for the transferred charge (5.18). This yields the following closed-form solution for q_d, the charge transferred under current-based timing.

$$\begin{aligned} q_d(i_d, V, i_0, R) =& -\frac{V}{R} T_d(i_d) - \frac{L(V - i_0 R)}{R^2}\left[\frac{V - i_d R}{V - i_0 R} - 1\right] \\ =& \frac{LV}{R^2} \log\left(\frac{V - i_d R}{V - i_0 R}\right) + \frac{L(i_d - i_0)}{R} \end{aligned} \tag{5.21}$$

Remark 5.7 (Monotonicity of charge per phase)**.** *Under current-based actuation, higher voltages lead to shorter timings. This can be seen from* (5.20)*. Even though the transferred charge q from* (5.18) *increases with V, combined with timing, the charge transferred in a phase (q_d from* (5.21) *with parameters from Table 5.2) decreases with increasing V for both phases. The following calculations yield $\frac{\mathrm{d}}{\mathrm{d}V} - q_t \leq 0$ and $\frac{\mathrm{d}}{\mathrm{d}V} q_r \leq 0$ to demonstrate this. To that end, they use $\log(\frac{1}{v}) = -\log(v)$ and $\log(1+v) \geq \frac{v}{1+v} \quad \forall v > -1$.*

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}V}\frac{R_t^2}{L}(-q_t) =& -\left[\log\left(\frac{V - RI}{V}\right) + V\frac{V}{V - RI}\cdot\frac{RI}{V^2}\right] = \frac{-RI}{V - RI} - \log\left(1 - \frac{RI}{V}\right) \\ \leq& \frac{-RI}{V - RI} - \frac{-RI}{V} \Big/ \frac{V - RI}{V} = 0 \\ \frac{\mathrm{d}}{\mathrm{d}V}\frac{R_r^2}{L} q_r =& \log\left(\frac{V}{V + RI}\right) + V\frac{V + RI}{V}\cdot\frac{RI}{(V + RI)^2} = -\log\left(1 + \frac{RI}{V}\right) + \frac{RI}{V + RI} \\ \leq& -\frac{RI}{V} \Big/ \frac{V + RI}{V} + \frac{RI}{V + IR} = 0 \end{aligned} \tag{5.22}$$

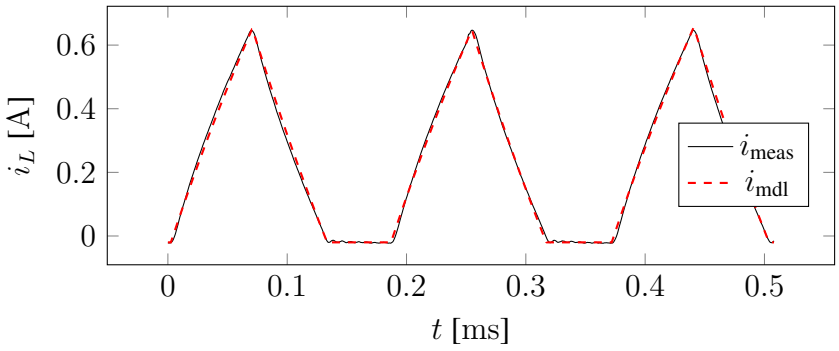


Figure 5.18: *The inductor current i_L from the model* (5.17) *remains close to a corresponding measurement. Note how the diode blocks the current as soon as it reaches $i_L = 0$ in this experiment.*

Remark 5.8 (Maximum current)**.** *Inductor current $i(t)$ from* (5.17) *converges against a maximum over time.*

$$\lim_{t\to\infty} i(t) \to i_\infty = \frac{V}{R}$$

This value is important during charging phase ϕ_t where it represents an upper bound for the user-selected current $i_d = I$. Eq. 5.20 for T_d reflects this bound. The logarithm, defined only for positive inputs, fails to evaluate there when $i_d > \frac{V}{R}$ and $i_0 = 0$ because $(V - \frac{V}{R}R)/V = 0$.

Validation of constant voltage dynamics The behavior of the closed-form model from this section has been compared to measurements on the demonstrator platform as well as small test setups. Fig. 5.18 shows an example from this measurement series with inductance $L = 290\,\mu\text{H}$, diode voltage $V_d = 0.65\,\text{V}$ and resistances $R_t = R_r = 1.15\,\Omega$. The circuit has been actuated using two phases with fixed timing and diode routing during ϕ_r, i.e., using the alternative for experiments described in Section 5.5.1. The voltages were measured to be $V_t = 3.26\,\text{V}$ and $V_r = 2.09\,\text{V}$ for transmitting and receiving cell, respectively.

The model current in Fig. 5.18 matches the measurement well. Note that the elevated resistance in this experiment leads to imperfect triangles where the rising edge curves outwards and the falling edge curves inwards. This curvature is caused by the dissipative losses and is well captured by the model.

As the voltages are measured and taken as constant in this experiment, we cannot derive statements about the battery model from these measurements. They motivate, however, that assuming constant voltages inside the brief switching phases is a reasonable approach. A more extensive validation of transfer models can be found in Fig. 5.20 and Fig. 5.21 (Section 5.8.3).

5.8 Large-scale Active Cell Balancing (ACB) simulation

After the description of battery models, ACB actuation, and intra-phase dynamics, this section combines these collected insights to achieve accurate, but fast simulation of large ACB scenarios. To that end, it presents and compares four simulation approaches that build on one

another. The first uses a numerical solver to simulate the intra-phase dynamics directly (Section 5.8.1). This reference solution represents a general purpose circuit simulator and is too slow to be useful for most applications. The second approach (Section 5.8.2) uses the constant voltage dynamics from Section 5.7 to obtain an iterative method. This is faster than the first approach because it forgoes the numerical solver but can still require one hour of computation time for large balancing scenarios. Instead of calculating that iteration phase by phase, the third approach (Section 5.8.3) aggregates phases by applying error control and adaptive step size techniques from the ODE domain. This systematically reduces millions of phase evaluations that would be necessary otherwise to dozens in some cases and enables interactive applications. While these approaches remain quite flexible in terms of actuation method and battery model, further speedup is possible with fixed-timing actuation and a simple battery model. The fourth approach (Section 5.8.4) examines this situation and finds a solution to the recursion which provides instant evaluation.

5.8.1 Straightforward numerical solution

The equivalent circuit models from Section 5.3 with a battery model like the ones presented in Section 5.4 are sufficient to simulate charge transfers using a general purpose numerical solver. While this approach is not computationally efficient, it serves as accuracy as well as computation time reference for later techniques and summarizes the insights obtained there.

This section assumes a battery model with two RC stages which is the most common size, but more stages can be appended as needed. Fig. 5.7 from Section 5.4 shows the equivalent circuit of the transfer to be simulated. The corresponding dynamics are in (5.1) and (5.5) which altogether yield the following ODE system.

$$\frac{\mathrm{d}}{\mathrm{d}t} i = \frac{1}{L}[V - iR] \qquad \frac{\mathrm{d}}{\mathrm{d}t} Q_0 = -i \tag{5.23}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} Q_1 = -i - \frac{V_1(Q_1)}{R_1} \qquad \frac{\mathrm{d}}{\mathrm{d}t} Q_2 = -i - \frac{V_2(Q_2)}{R_2} \tag{5.24}$$

These system dynamics have four states per cell and 8 states in total: i_t, i_r are the respective cell currents, whose direction follows the convention from Definition 5.1, and $Q_{t,0}, \ldots, Q_{r,2}$ are the charges in the respective RC stages. Q_0, a short notation for both $Q_{t,0}$ and $Q_{r,0}$, models the charge of the cell itself and thus the SoC. V, the respective cell voltage, is given by summing up the contributions from the involved stages (see also Fig. 5.7 from Section 5.4).

$$V_t = \sum_{j=0}^{2} V_{t,j}(Q_{t,j}) \qquad V_r = \sum_{j=0}^{2} V_{r,j}(Q_{r,j}) + V_d \tag{5.25}$$

V_d models additional voltage from the diode preventing undesirable discharge of c_r during ϕ_r or a potential ϕ_b as explained in Section 5.5.1. If such a diode is not involved, $V_d = 0$ should be used.

The following description mainly treats fixed-timing actuation with two phases. While an additional ϕ_b can be added analogously, the adjustments for fixed-current actuation are pointed out along the way. Given timing parameters T_t, T_c, a numerical solver can simulate the system as follows.

ϕ_t Solve ODE system (5.23) & (5.24) with $i_t(0) = 0$ and end time T_t to update transmitting cell c_t. This updates states $\begin{bmatrix} Q_{t,0} & Q_{t,1} & Q_{t,2} \end{bmatrix}$ and yields peak current $I = i_t(T_t)$. Simultaneously, relax receiving cell c_r by solving ODE (5.24). This updates states $\begin{bmatrix} Q_{r,1} & Q_{r,2} \end{bmatrix}$.

ϕ_r After the circuit switches over, relax transmitting cell c_t with ODE (5.24), updating $\begin{bmatrix} Q_{t,1} & Q_{t,2} \end{bmatrix}$. Solve ODE system (5.23) & (5.24) with $i_r(T_t) = -i_t(T_t)$ and end time T_c for c_r. This updates $\begin{bmatrix} Q_{r,0} & Q_{r,1} & Q_{r,2} \end{bmatrix}$.

* Repeat until K, the desired number of cycles, is reached.

Note that during ϕ_r, receiving time T_r is not known a priori. Its calculation can be circumvented by setting $i_r(t) := \max(0, i_r(t))$ before updating charges. Alternatively, one may use (5.20) from Section 5.7 to calculate the corresponding timing directly or perform a binary search over the time domain if the ODE solver permits. These alternative approaches also handle requirements like constant peak current I, implying T_t adjustments at runtime and immediate switching back to ϕ_t when $i = 0$ and T_r has elapsed to ensure $T_c = T_t + T_r$.

Switching losses, as described in Section 5.6, can be simulated with this approach by updating the states with the terms in (5.14) after each completed cycle.

5.8.2 Iterative solution for transfer dynamics

The first improvement over the direct application of numerical solvers for ACB simulation are phase-based models, like the one from Section 5.7. With a closed-form equation for the evolution during each phase they are significantly faster than state-of-the-art circuit simulators. As all capacitors in current cell models like [45] are large, modeling effects in the domain of seconds and minutes, treating their voltages as constant during individual phases does not introduce noticeable errors.

The formulas from Section 5.7 are only valid as long as the system configuration does not switch. Over one cycle of charge and discharge phase the dynamics move through different equivalent circuits with different parameters and initial conditions. For longer simulations with many cycles, an iterative approach is hence required. It is described in the following for the T-interface (Section 5.5.1).

1. Obtain cell voltages from (5.25).

2. Calculate peak current I given transmitting time T_t with (5.17).

3. Update charges for phase ϕ_t, using partially specialized $q_t(T) := q(T, V_t, 0, R_t)$ from (5.18) and $q_{t,j}$, $q_{r,j}$ from (5.19).

$$\begin{aligned} Q_{t,0}(t + T_t) &= Q_{t,0}(t) + q_t(T_t) \\ Q_{t,j}(t + T_t) &= Q_{t,j}(t) + q_t(T_t) + q_{t,j}(T_t, Q_{t,j}(t)) \\ Q_{r,0}(t + T_t) &= Q_{r,0}(t) \\ Q_{r,j}(t + T_t) &= Q_{r,j}(t) + q_{r,j}(T_t, Q_{r,j}(t)) \end{aligned} \tag{5.26}$$

4. Calculate break current I_b given transmitting time T_r with (5.17) (Fig. 5.10). Update charges for phase ϕ_r analogously to ϕ_t.

5. From timing formula (5.20), calculate discharging time T_b until $i = 0$ and discharge of c_r ends in phase ϕ_b. Update charges analogously to ϕ_t. Relax the RC stages for $T_c - (T_r + T_t)$ with q_j from (5.19).

6. Repeat, until K, the desired number of cycles, is reached.

The two-phase approach using a diode for the entire ϕ_r forgoes Step 4 and sets $I_b = I$ in Step 5, relabeling ϕ_b to ϕ_r. Additional changes are also necessary to simulate the I-interface (Section 5.5.2). In Step 2, T_t is calculated given I with timing formula (5.20). Step 4 is omitted, as for the two-phase approach, but letting $V_d = 0$ since no diode is used and adjusting $T_c = T_t + T_r$ dynamically since no break phase is included.

Switching losses (Section 5.6) are even easier to integrate with an iterative approach than with a numerical solver as in Section 5.8.1. Exchanging $\tilde{q}_t(T_t) := q_t(T_t) - q_{sw,t}$ and $\tilde{q}_r(T_r) := q_r(T_t) - q_{sw,r}$ for $q_t(T_t)$ and $q_r(T_r)$, respectively, is sufficient.

This approach is significantly faster than solving directly although it does not introduce additional errors. Please refer to Fig. 5.21 in Section 5.8.3 for a more extensive evaluation.

It is worth mentioning that this iteration cannot be replaced by a matrix exponentiation in general. The main obstacle is the voltage dependency of the charge differences q in the I-interface. This dependency leads to a system of the form $x[k+1] = A(x[k])x[k]$ where A depends on the state in a nonlinear fashion. A special case for the T-interface is explored in Section 5.8.4.

5.8.3 Error-controlled, adaptive phase aggregation

This section transform the iterative approach from the previous section and then applies error control methods from the ODE domain for faster computation. To reformulate the iteration, we now consider the evolution with respect to cycles k in contrast to previous sections which calculate the system dynamics over time t. Each cycle has a duration of T_c that varies slightly if a constant peak current is maintained as voltages change. It is thus necessary to track time as a separate state variable in this formulation. Tracking time in this way is simpler than scaling results to account for time variations. This perspective also makes it easier to reason about the accuracy loss outside cycle ends that we accept to alleviate kinks (see Fig. 5.19).

In the following, we adopt the formulations from Section 5.7 to sum up the state differences

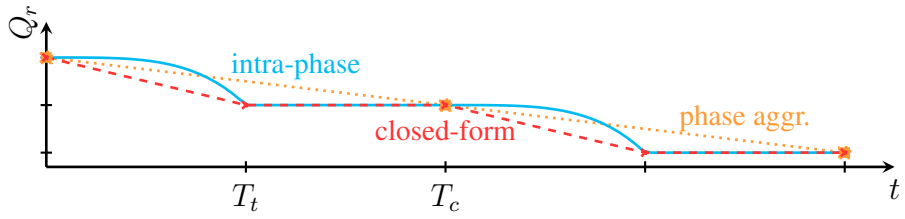


Figure 5.19: *Modeling intra-phase behavior (—, Section 5.8.1) provides the most accurate charge information at all times. When solving phases in closed form (- - -, Section 5.8.2), we obtain accurate charge information only at the end of phases. In order to avoid kinks, the model underlying phase aggregation (·····, Section 5.8.3) is content being accurate only at the end of cycles.*

of a single cycle:

$$\begin{aligned}
\begin{bmatrix} V_t & V_r \end{bmatrix} &= \begin{bmatrix} \sum_{j=0}^{2} V_{t,j}(Q_{t,j}) & \sum_{j=0}^{2} V_{r,j}(Q_{r,j}) + V_d \end{bmatrix} \\
\begin{bmatrix} I & T_r \end{bmatrix} &= \begin{bmatrix} i(T_t, V_t, 0, R_t) & T_d(0, V_r, -I, R_r) \end{bmatrix} \\
\frac{\Delta t}{\Delta k} &= T_c \\
\frac{\Delta Q_t}{\Delta k} &= q(T_t, V_t, 0, R_t) \\
\frac{\Delta Q_{t,j}}{\Delta k} &= q(T_t, V_t, 0, R_t) + q_{t,j}(T_c, Q_{t,j}) \quad , j = 1, 2 \\
\frac{\Delta Q_r}{\Delta k} &= q(T_r, V_r, -I, R_r) \\
\frac{\Delta Q_{r,j}}{\Delta k} &= q(T_r, V_r, -I, R_r) + q_{r,j}(T_c, Q_{r,j}) \quad , j = 1, 2
\end{aligned} \tag{5.27}$$

This is the formulation for two phases with fixed timing parameters T_t, T_c. A third phase or switching losses can be integrated as in Section 5.8.2. Constant peak current (I-interface) can be achieved by calculating T_t, T_c on the fly with (5.20) in addition to T_r.

(5.27) is not inherently an ODE because $k, \Delta k \in \{1, 2, \dots K\}$ are discrete variables. Nevertheless, we are interested in applying ODE techniques with error control and adaptive step size. The typical choice for these requirements are solvers from the class of embedded Runge-Kutta methods. An unadjusted solver from that class may also evaluate the right-hand side for non-integral cycle counts and thus at instants where charge evolution is incorrectly interpolated. We will first quantify the magnitude of this issue and then propose a remedy for it. Consider the following worst case calculation. With an average balancing current equivalent to a C rate of 1 and using a low frequency with $T_t = 10\text{ms}$, the SoC changes only by

$$\Delta z = \frac{\Delta Q}{C_0} = \frac{\frac{1 \cdot C_0}{3600\text{s}} \cdot 10\text{ms}}{C_0} \approx 2.7\text{e-}4\text{pp} = 0.027\text{bp}.$$

Even assuming a region where the OCV increases quickly by $100\frac{\text{mV}}{\text{pp}}$, Δz corresponds to only $\Delta V = 0.027\text{mV}$. With a minimum cell voltage of 2.5V, the worst case error thus remains in the

order of 1e-5 and is typically far smaller. As cell models come with inherent modeling errors in the order of 1e-3 [45], one may accept this error and resort to readily available methods directly.

Alternatively, ensuring that embedded Runge-Kutta solvers only evaluate integral cycle counts is also possible. These solvers have fixed nodes, proportional to the current step size Δk, at which a function is evaluated. For instance, the Bogacki-Shampine method [22], combining orders 2 and 3, evaluates at $0\Delta k$, $\frac{1}{2}\Delta k$, $\frac{3}{4}\Delta k$ and $1\Delta k$. Restricting steps Δk to multiples of their common denominator, here 4, ensures that all evaluation points are integral. Other popular embedded methods with orders 4 & 5, like Dormand-Prince [56] which is the default solver in MATLAB and Cash-Karp [33] similarly require multiples of 90 and 40, respectively.

Experimental Results

To evaluate the approaches from the previous sections, they have been implemented with the C++ library boost::odeint [134]. This forms the back end of a Python implementation that we use for data analysis.

All battery parameters correspond to the $850\,\mathrm{mA}$ Li-Ion polymer cell which has been characterized in [45] for various SoC levels. The analytic OCV relation there is replaced by a piecewise linear formulation because that interacts better with the rest of our framework. This model comes with a good SoC (or runtime) error of $0.4\,\%$ and a voltage deviation of $15\,\mathrm{mV}$.

We evaluate accuracy and speedup of the proposed techniques in two stages. First, we compare the intra-phase model from Section 5.8.1 to a circuit simulator to quantify the errors introduced by the equivalent circuit abstraction. Next, we compare the abstraction levels from Sections 5.8.1, 5.8.2, and 5.8.3 to each other, using longer simulation times of several minutes. Although impossible for the circuit simulator, such times are still small for ACB strategies.

Equivalent circuit model accuracy & speedup We investigate a transfer between a transmitting cell at $60\,\%$ SoC and a receiving cell at $40\,\%$, setting the cell parameters accordingly. In addition, we modeled an inductor resistance $R_L = 1\,\mathrm{m\Omega}$ and a diode voltage $V_d = 0.7\,\mathrm{V}$. This transfer is simulated in SPICE and with the intra-phase ODE from Section 5.8.1. We perform the transfer over a variety of switching frequencies, keeping the peak current approximately constant by suitably decreasing the inductance with shorter timing. All transfers are

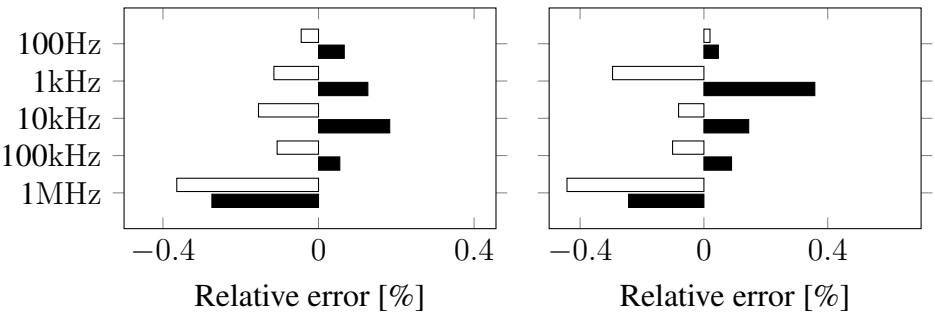


Figure 5.20: *For both nominal currents of* $400\mathrm{mA}$ *(left) and* $2\mathrm{A}$ *(right), the relative error in SoC of transmitting (black) and receiving (white) cell remains small compared to SPICE over a wide frequency range.*

executed over 10 cycles of fixed timing. Fig. 5.20 shows two sets of this experiment. The nominal peak currents, assuming lossless dynamics, were $400\,\mathrm{mA}$ and $2\,\mathrm{A}$. Although we do not see a discernible trend, the relative error in SoC remained below $0.5\,\%$ over the investigated frequencies. We have selected these frequencies to span more than the most relevant range of 1–$100\,\mathrm{kHz}$. The SoC evolution depends on all voltages and is, as such, the most error-sensitive value in the system. Relative errors in the various voltages all remained even smaller. We attribute these errors to several aspects that are only modeled in the circuit simulator, like diodes and switches or imperfect, but realistic current edges. Since the errors do not exceed inherent modeling errors of the cell model, the equivalent circuit abstraction is well justified.

The speedup in this experiment is significant. All SPICE simulations required over $5\,\mathrm{s}$ to calculate 10 cycles at the desired accuracy. With the equivalent circuit abstraction, the intra-phase model solves the same task in the millisecond range. Although the simulation time varies widely, the higher frequency and accuracy requirement in the shorter experiments keep the computation effort roughly constant.

Abstraction level accuracy & speedup The previous paragraphs evaluate inaccuracies introduced by the equivalent circuit abstraction. Now, we investigate the techniques from Sections 5.8.1, 5.8.2, and 5.8.3 which build on that abstraction. For this purpose, we consider the same cell pair with $40\,\%$ and $60\,\%$ SoC over longer simulation times with randomized transfer parameters. We draw resistances $R = 10^x\,\Omega, x \in [-3, 0]$ and peak currents $I \in [0.1\,\mathrm{A}, 2.5\,\mathrm{A}]$, according to a uniform distribution. Timing parameters were fixed to $\begin{bmatrix} T_t & T_c \end{bmatrix} = \begin{bmatrix} 200\,\mu\mathrm{s} & 420\,\mu\mathrm{s} \end{bmatrix}$. The inductance $L = \frac{4.0\,\mathrm{V}}{I} T_t$ is scaled to approximately achieve the desired peak current. After drawing the random parameters, we perform $K = 10^6$ cycles with the respective settings.

Fig. 5.21 shows the results from 50 such random transfers. Although the closed form approach from Section 5.8.2 and the phase aggregation technique from Section 5.8.3 introduce *virtually no additional error*, they achieve a *remarkable speedup*. The closed form approach is roughly 35 times faster than the intra-phase ODE solver, and phase aggregation is another 2500 times faster. Compared to intra-phase ODE, phase aggregation hence achieves a speedup

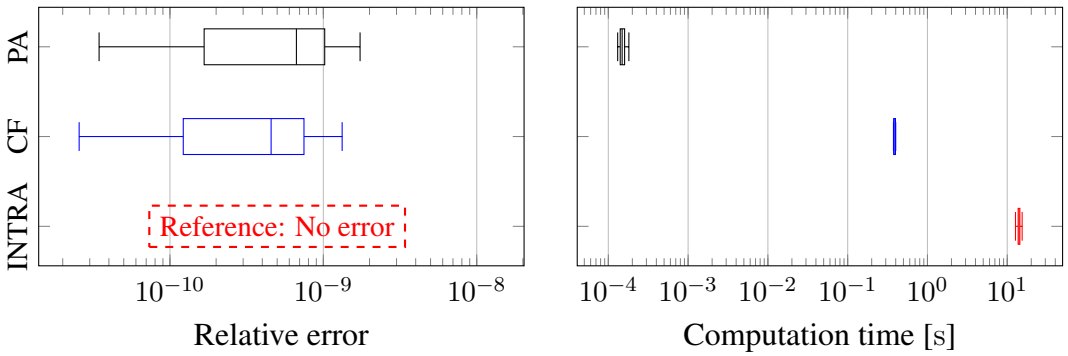


Figure 5.21: *Neither closed form (CF, Section 5.8.2) nor phase aggregation (PA, Section 5.8.3) method introduce noticeable errors into the simulation over a direct solution of the equivalent circuit dynamics (INTRA, Section 5.8.1). At the same time, they do achieve speedups of several orders of magnitude.*

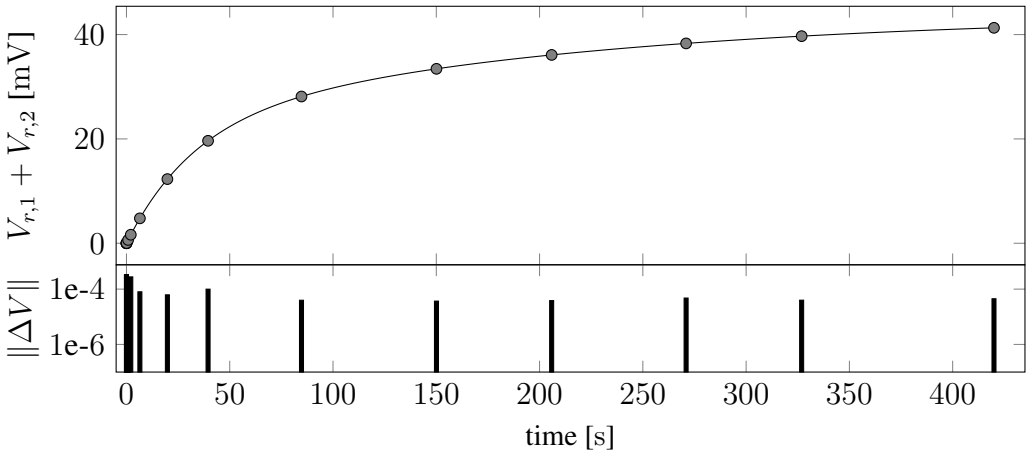


Figure 5.22: *The phase aggregation approach (circles) tracks the intra-phase ODE solution (line) perfectly, calculating only dozens, not millions, of cycles. In the RC voltage of the receiver, the absolute error remains around* $0.1\,\mu\mathrm{V}$.

of about 90000. These measurements were made on a workstation with $3.4\,\mathrm{GHz}$ Intel i7-3770 CPU and $16\,\mathrm{GB}$ RAM. All implementations are single-threaded. Note that the computation time of each approach does not vary a lot over the experiment set, indicating that the required effort depends on the number of simulation cycles and not the total simulation time. With higher frequency, the proposed methods therefore become even more beneficial.

Extrapolating the computation times from Fig. 5.21 involving two cells and a simulation time of about 7 minutes, we consider a battery pack that performs 10 parallel transfers, using a frequency of $20\mathrm{kHz}$ (10 times higher) and a simulation time of 7 hours. For this 6000 times larger scenario, a general purpose solver requires at least a full day. Closed-form iteration reduces that time, but still needs more than 40 minutes. Only the phase aggregation approach calculates results in less than 2 seconds and remains fast enough to interactively evaluate balancing strategies with respect to balancing time and efficiency.

Fig. 5.22 shows an example time series plot from one of the random transfers that we performed, demonstrating why phase aggregation is so much faster. Instead of calculating millions of cycles one by one, it requires only dozens of evaluations to provide the desired result. This plot also demonstrates the importance of the RC stages in the model, as the voltage they contribute on the receiver side exceeds $40\mathrm{mV}$.

5.8.4 Long-term charge transfer simulation with fixed timing

Even though the phase aggregation approach from Section 5.8.3 leads to rapid simulations, it is possible to be even faster in certain cases. This section presents such a case that assumes (i) the simulation remains in one linear segment of the charge-voltage mapping from Section 5.4, (ii) the constant voltage argument from Section 5.7 holds, (iii) the T-interface is used, and (iv) the current is low enough that RC stages need not be modeled. (i) and (ii) are almost always satisfied as explained in the respective sections. (iii) is almost entirely a preference of the system designer. For (iv), the voltage contributions can be estimated from the expected current levels

using (5.7). Please also refer to the discussion there.

Sending cell voltage evolution

We begin with the easier case of the transmitting cell. Take one segment from the piecewise linear charge-voltage mapping (5.8). We substitute this segment along with the parameters of the sender into Eq. (5.18), the charge transferred according to the first order model. This yields recurrence relation

$$\begin{aligned} V_t[k+1] =& V_t[k] + \zeta q_t(V_t[k]) \\ =& \Bigg[\underbrace{1 - \left(\zeta\frac{T_t}{R_t} + \zeta\frac{L}{R_t^2}\left(e^{-R_tT_t/L} - 1\right)\right)}_{=:\alpha}\Bigg] V_t[k]. \end{aligned} \tag{5.28}$$

As T_t remains constant during Δt, the newly defined $\alpha \in \mathbb{R}$ is also constant there. Given α, it is trivial to see that we can calculate $V_t[k]$ for any number of switching cycles k as

$$V_t[k] = \alpha^k V_t[0]. \tag{5.29}$$

Receiving cell voltage evolution

Calculating the receiving side is more intricate because it depends on the evolution of the sending cell. The peak current decreases as the voltage of the sender decreases during a transmission and this must be taken into account. The biggest challenge from a mathematical point of view is the charge transferred over the MOSFET diode during T_b (see Figure 5.10 on page 94) because it introduces non-linear terms. This term is usually very small compared to the charge received directly over a conducting MOSFET during T_r. We hence ignore it at first and then adjust our calculation at the end of this section. We can hence combine the charge of a single cycle from Eq.(5.18) for T_r and the cell voltage model (5.8) to describe the major behavior.

$$\begin{aligned} V_r[k+1] =& V_r[k] + \zeta q_r(V_r[k], i_0) \\ =& V_r[k] + \zeta\left[\frac{-V_r[k]T_r}{R_r} - \frac{L(V_r[k] + IR_r)}{R_r^2}\left(e^{-R_rTr/L} - 1\right)\right] \\ =& \Bigg[\underbrace{1 - \zeta\frac{T_r}{R_r} - \zeta\frac{L}{R_r^2}\left(e^{-R_rTr/L} - 1\right)}_{=:\beta}\Bigg] V_r[k] - \zeta\frac{LI}{R_r}\left(e^{-R_rTr/L} - 1\right) \end{aligned} \tag{5.30}$$

This recurrence relation for V_r depends on V_t through I, the varying peak current at the end of the charging phase. Consider that current as given by Eq. (5.17) and introduce another helper constant $d \in \mathbb{R}$ as follows.

$$I = \frac{1}{R_t}\left(1 - e^{-R_tT_t/L}\right) V_t[k] =: dV_t[k] \tag{5.31}$$

Given the current in this form, we can now further transform Eq. (5.30).

$$V_r[k+1] = \beta V_r[k] + \underbrace{\zeta \frac{-L}{R_r}\Big(e^{-R_r T_r/L} - 1\Big)}_{=:\theta} d\, V_t[k] \tag{5.32}$$

Now, the recurrence relation for V_r can be solved with the following lemma.

Lemma 5.9 (Recurrence relation with offset)**.** *Consider a recurrence relation*

$$x_2[k+1] = \theta \alpha^k x_1[0] + \beta x_2[k] + \gamma \tag{5.33}$$

with given parameters $\alpha, \beta, \gamma, \theta, x_1[0], x_2[0] \in \mathbb{R}$ and $\alpha \neq \beta$. If $\beta \neq 1$, then $x_2[k]$ has the following unique solution.

$$x_2[k] = \beta^k \big(x_2[0] + \frac{\theta}{\beta - \alpha} x_1[0]\big) + \frac{\beta^k - 1}{\beta - 1}\gamma - \frac{\theta \alpha^k}{\beta - \alpha} x_1[0]. \tag{5.34}$$

Proof. Introduce auxiliary variable $y[k]$ with

$$y[k] := x_2[k] + \frac{\theta \alpha^k}{\beta - \alpha} x_1[0]. \tag{5.35}$$

$y[k]$ has a simpler recursion than $x_2[k]$. This can be seen as follows.

$$\begin{aligned} y[k+1] =& x_2[k+1] + \frac{\theta \alpha^{k+1}}{\beta - \alpha} x_1[0] = \theta \alpha^k x_1[0] + \beta x_2[k] + \gamma + \frac{\theta \alpha^{k+1}}{\beta - \alpha} x_1[0] \\ =& \theta \alpha^k x_1[0] + \beta \Big(y[k] - \frac{\theta \alpha^k}{\beta - \alpha} x_1[0]\Big) + \gamma + \frac{\theta \alpha^{k+1}}{\beta - \alpha} x_1[0] \\ =& \beta y[k] + \gamma + \theta \alpha^k x_1[0] \Big\{ \underbrace{1 - \frac{\beta}{\beta - \alpha} + \frac{\alpha}{\beta - \alpha}}_{=0} \Big\} \end{aligned}$$

Here, the first step uses the definition of $y[k]$ in (5.35). Next, the recurrence relation for $x_2[k+1]$ from (5.33) is substituted. Then, $y[k]$ is recovered from $x_2[k]$ via its definition in (5.35) ($x_2[k] = y[k] - \frac{\alpha^k \theta}{\beta - \alpha} x_1[0]$). After these steps, the recursion of $y[k]$ is solved by

$$y[k] = \beta^k y[0] + \frac{\beta^k - 1}{\beta - 1}\gamma. \tag{5.36}$$

This can quickly be confirmed by induction over k.

Given the original recursion from (5.33), we can therefore initialize $y[0] = x_2[0] + \frac{\theta \alpha^0}{\beta - \alpha} x_1[0]$ and substitute it into (5.36). Transforming that equation back to $x_2[k]$ using (5.35) as in previous steps, yields (5.34) and concludes the proof. □

Note that Lemma 5.9 requires $\beta \neq \alpha$ which follows from $R_t < R_r$ in most cases. If required, it can even be enforced by slightly altering T_r, i.e., by transferring slightly more charge over the diode at the end of the discharge phase (cf. Figure 5.10).

We now have all components to run simulations concerning the major phases ϕ_t and ϕ_r. (5.29) models long-term evolution for the sender, (5.32) with Lemma 5.9 for the receiver. Required helper variables, comprising equivalent circuit resistances R_t, R_r, inductor timings T_t, T_r, inductance L, and voltage slope ζ, are summarized in (5.37).

$$\begin{aligned} \alpha =& 1 - \left(\zeta\frac{T_t}{R_t} + \zeta\frac{L}{R_t^2}\left(e^{-R_tT_t/L} - 1\right)\right) & \gamma =& 0 \\ \beta =& 1 - \zeta\frac{T_r}{R_r} - \zeta\frac{L}{R_r^2}\left(e^{-R_rTr/L} - 1\right) & \theta =& \zeta\frac{-L}{R_r}\left(e^{-R_rT_r/L} - 1\right)d \end{aligned} \tag{5.37}$$

Here, d describes the peak current evolution $i_0 = dV_t[k]$, as defined in (5.31).

The following paragraphs improve simulation accuracy by considering the switching effects described in Section 5.6 and the diode involvement explained in Section 5.5.1.

Switching losses for long-term transfer with fixed timing

Consider the textbook switching losses in charge from from Eq. (5.14). With this, the cell evolutions from (5.28) and (5.32) adjust to the following.

$$\begin{aligned} V_t[k+1] =& \alpha V_t[k] - \frac{\zeta}{2}\tau_d \underbrace{dV_t[k]}_{=I} - \frac{\zeta}{2}C_{OSS}V_t[k] \\ V_r[k+1] =& \beta V_r[k] + \theta V_t[k] - \frac{\zeta}{2}\tau_u \underbrace{dV_t[k]}_{=I} - \frac{\zeta}{2}C_{OSS}V_r[k] \end{aligned} \tag{5.38}$$

The recursion parameters from (5.37) for Lemma 5.9 must hence be adjusted as follows.

$$\begin{aligned} \tilde{\alpha} :=& \alpha - \frac{\zeta}{2}\left[C_{OSS} + \tau_d d\right] & \tilde{\gamma} :=& \gamma \\ \tilde{\beta} :=& \beta - \frac{\zeta}{2}C_{OSS} & \tilde{\theta} :=& \theta - \frac{\zeta}{2}\tau_u d. \end{aligned} \tag{5.39}$$

With these adjustments, the simulation proceeds as it did previously without switching losses. Eq. (5.29) calculates the sender voltage for the macro step Δt after calculating the necessary cycles $K = \frac{\Delta t}{T_t+T_r+T_b}$. Lemma 5.9 subsequently yields the resulting receiver voltage.

Receiving over diode

The previous paragraphs have not yet dealt with q_b, the charge transferred during break ϕ_b (see also Fig. 5.10 in Section 5.5.1). Since T_b is typically short, the error thus created may be acceptable in many cases. It is possible, however, to improve this error considerably by using a rough estimate for q_b.

q_b can be calculated accurately using q_d from (5.21). Using this formula, that substitutes timing into the charge term to obtain the transferred amount of a specific phase, we derive

$$q_b = \frac{-L(V_r + V_d)}{R_r^2} \log\Big(\frac{(V_r + V_d) - 0R_r}{(V_r + V_d) + I_b R_r}\Big) + \frac{L(-I_b - 0)}{R_r} \tag{5.40}$$

Since I_b varies over time, this precise version of q_b cannot be combined directly with the linear models from the previous paragraphs.

As an alternative, we hence consider a coarse estimate for q_b instead. If we consider the lowest peak current from the last cycle K, $I_{\text{min}} = i(T_t, V_t[K], 0, R_t) = dVt[K]$, and a constant receiver voltage, $V_r = V_r[0]$, we receive a constant proxy value $I_{b,p}$ for I_b.

$$I_{b,p} = i(T_r, V_r[0], -I_{\text{min}}, R_r) = \frac{V_r[0]}{R_r}\big(1 - e^{R_r T_r/L}\big) - dV_t[K]e^{R_r T_r/L} \tag{5.41}$$

Here, i is the current from Eq. (5.17), obtained as unique solution for the intra-phase dynamics. Inserting $I_{b,p}$ and $V_b[0] = V_r[0] + V_d$ into q_b from (5.40) then yields $q_{b,p}$, the required proxy for q_b. To take this constant value into account in the previous calculations, only γ from (5.37) needs to be adjusted.

$$\hat{\gamma} := \gamma + \zeta q_{b,p} \tag{5.42}$$

The improvement this correction term brings is evaluated at the end of this section (Table 5.3). Even more precise results may be obtainable by calculating $V_r[K]$ without diode correction at first and then adjusting it afterwards, potentially with an iterative approach. Another alternative could use error control as in Section 5.8.3 specifically for this diode correction term. On the other hand, the voltage evolves slowly even during a typical macro step period $T_m \approx 1\,\text{min}$ and the share of charge transferred during T_b is small. For this reason, the increased computational effort of more complicated techniques is typically not justified.

Smart cell simulation platform

The fixed timing interface (Section 5.5.1) is the easiest to implement and is hence the most likely to be deployed on embedded platforms. In addition, the calculation methods of this section provide the most accurate instant evaluation. For this reason, they form the battery layer in the co-simulation framework developed at TUM CREATE. Together with a CAN bus model for the communication layer, this tool allows quick evaluation of request-driven strategies for balancing in a distributed BMS. It has demonstrated, for instance, that the bus utilization remains low in a smart cell setup and there is no communication bottleneck for the investigated strategies. Neither cell communication nor request-driven strategies have consequently been a focus of this thesis. A screenshot of the GUI front end of the smart cell simulation tool is shown in Fig. 5.23, visualizing the SoC evolution over time during an active cell balancing run of a 96 smart cell battery pack.

Long-term transfer model evaluation

In the following, we examine accuracy and speedup that the simulation approach for the T-interface from this section achieves. As the computation effort remains constant with respect

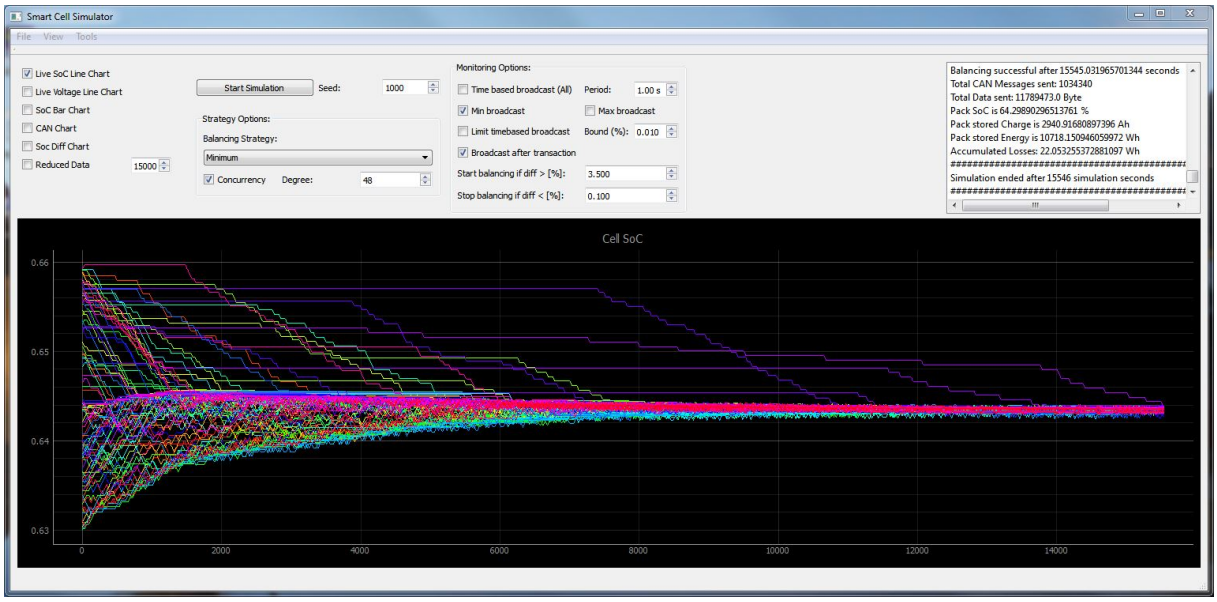


Figure 5.23: *Co-simulation framework performing simulation of active cell balancing for the 96 smart cells of a* $21.6\,\mathrm{W\,h}$ *battery pack.*

to the phase count, we expect a linear speedup over the iterative approach from Section 5.8.2 which calculates phase by phase. On the other hand, we also expect the inaccuracies from the diode involvement in phase ϕ_b to grow as more cycles are calculated.

In the first experiment, we analyze a set of transfer scenarios and compare the results from "normal" $\mathrm{LiFePO_4}$ cells with $1.1\,\mathrm{A\,h}$ (Fig 5.8) and a "tiny" variant at $0.1\,\mathrm{A\,h}$ with the same voltage curve. The tiny cells are chosen to evaluate the effects from somewhat faster voltage evolution. The scenarios are created by considering every combination of the following parameter vectors individually.

$$\begin{aligned} I &\in \{0.25, 1.0, 2.0\} & \begin{pmatrix} R_s & R_r \end{pmatrix} &\in \left\{ \begin{pmatrix} 0.01 & 0.012 \end{pmatrix}, \begin{pmatrix} 0.5 & 0.6 \end{pmatrix}, \begin{pmatrix} 1.3 & 1.4 \end{pmatrix} \right\} \\ L &= 0.0001 & \begin{pmatrix} z_s[0] & z_r[0] \end{pmatrix} &\in \left\{ \begin{pmatrix} 0.4 & 0.3 \end{pmatrix}, \begin{pmatrix} 0.8 & 0.2 \end{pmatrix}, \begin{pmatrix} 0.45 & 0.65 \end{pmatrix} \right\} \end{aligned} \tag{5.43}$$

For a macro step size of $10\mathrm{s}$, driving the simulation in a phase-by-phase fashion requires around $7\mathrm{s}$ in a prototypic Python implementation. The reason is that $10\mathrm{s}$ of balancing time may correspond to over 10^5 switching cycles. By contrast, the approach from this section requires less than $100\mu\mathrm{s}$ on average, a speedup of roughly 10^5 for this macro step. The analysis of the worst case relative error $\epsilon_{\mathrm{rel}} := \frac{\|\Delta Q_{\mathrm{cf}} - \Delta Q_{\mathrm{step}}\|}{\|\Delta Q_{\mathrm{step}}\|}$ reveals, however, that changes in voltage lead to errors from the third phase ϕ_b. Without taking diode involvement into account, ϵ_{rel} is in the order of 10^{-3} for both cell variants. Using the correction term from Eq. (5.42), the relative error is reduced to $\epsilon_{\mathrm{rel}} \approx 10^{-5}$, the error of the tiny variant being about 5 times larger.

Cells with larger capacities lead to a more benign situation since the current is usually not raised equally for efficiency reasons and the voltages hence evolve more slowly. Overall, this technique is hence well-suited to calculate the transfer dynamics in one go, even for macro steps in the low minute range.

Another experiment investigates speedup and relative error for a small battery pack. It is made up of parallel-connected units with 10 $LiFePO_4$ 1.1 A h cells (Fig 5.8). Two such cell units then transfer charge between each other for 1 s at $I = 2.0$ A, using a variety of resistance and inductance parameters. Table 5.3 shows the results from this experiment. Although coarse, the diode correction term (5.42) improves accuracy by almost two orders of magnitude without significant impact on computation times. While it is intuitive that speedup increases and accuracy decreases with lower inductance and the implied higher frequency, the dependency on resistance may be worth investigating.

Overall, the technique is less accurate but somewhat faster than the error-controlled approach from Section 5.8.3. The decreased accuracy may not be relevant, however, since the largest errors are still introduced by the battery model and the equivalent circuit abstraction.

All measurements from this section have been performed on a laptop computer with Intel i5-2540M CPU @ 2.6 GHz and 8 GB RAM.

Table 5.3: *Over various inductance (vertical) and resistance (horizontal) values, the instantaneous evaluation of the fixed-timing approach is dramatically faster than the iteration from Section 5.8.2. Accuracy is helped significantly by the diode correction and remains well above the levels of the battery model.*

		Basic version			**WITH diode correction**		
		0.005	0.05	0.5	0.005	0.05	0.5
Speedup	$3.0e{-6}$	$1.0e{+5}$	$1.1e{+5}$	$1.2e{+5}$	$9.4e{+4}$	$9.5e{+4}$	$9.6e{+4}$
	$1.0e{-5}$	$3.8e{+4}$	$3.5e{+4}$	$3.4e{+4}$	$2.4e{+4}$	$3.0e{+4}$	$2.8e{+4}$
	$3.0e{-5}$	$8.4e{+3}$	$1.2e{+4}$	$8.8e{+3}$	$9.5e{+3}$	$9.3e{+3}$	$9.4e{+3}$
	$1.0e{-4}$	$3.4e{+3}$	$3.1e{+3}$	$3.4e{+3}$	$3.0e{+3}$	$2.8e{+3}$	$3.0e{+3}$
		0.005	0.05	0.5	0.005	0.05	0.5
Rel. Error	$3.0e{-6}$	$7.5e{-4}$	$7.4e{-4}$	$8.8e{-4}$	$1.8e{-5}$	$1.3e{-5}$	$2.2e{-5}$
	$1.0e{-5}$	$7.4e{-4}$	$7.5e{-4}$	$8.8e{-4}$	$4.0e{-6}$	$4.5e{-6}$	$3.0e{-6}$
	$3.0e{-5}$	$7.4e{-4}$	$7.5e{-4}$	$8.8e{-4}$	$1.6e{-6}$	$7.7e{-7}$	$2.0e{-6}$
	$1.0e{-4}$	$7.4e{-4}$	$7.5e{-4}$	$8.8e{-4}$	$1.3e{-7}$	$2.4e{-7}$	$2.1e{-6}$

5.9 Related work

In this section, the most relevant contributions from the domain of ACB are discussed. On the one hand, we deal with other modeling techniques. On the other hand, we go over the general context of BMS design and alternative ACB approaches.

Battery management Although Electrical Energy Storage (EES) already serves many use cases, upcoming technologies like Electric Vehicles (EVs) and smart grid applications would benefit the most from further improvements [151]. The smart grid concept aims to deliver electricity both cheaper and more reliably by augmenting the power grid with a communication network and sophisticated control. These efforts aim to deal with more and more irregularities in the grid that increasing amounts of renewable energy introduce, driven by ambitious targets like 33 % by 2020 in the European Union [175] or 15–30 % in about half of the US by 2025 [137].

The overall motivations, emission reduction and oil independence, extend to the transportation sector and also stimulate the EV development. As both domains employ ever larger EES devices, "a smart [Battery Management System (BMS)] is crucial [for their] realization" states Rahimi-Eichi [151] in his overview.

On the other hand, "there is still no consensus of the final definition of BMS and what [BMSs] do" [120]. Under the premise that a BMS is any system that manages the battery, Lu [120] presents the key issues of Li-Ion battery management, like the small size of the safe operating window or the difficulties in measuring SoC. The high-level tasks of a BMS identified in the recent works of Lu and Rahimi-Eichi, *monitoring, safety, thermal management, state estimation, and cell balancing*, are similarly discussed in earlier works like [70] or [87]. The corresponding engineering aspects are discussed more thoroughly in Andrea's book [6]. Besides control and power engineering, the embedded systems domain also has its own perspective on BMS implementation, presented in the brief overview by Brandl [27].

The following paragraphs expand on *state estimation and cell balancing* and their requirements like battery models. Please refer to the aforementioned works for a discussion on other BMS functions.

Decentralized implementation In commercial battery packs, centralized BMS architectures are currently dominating. Here, a central master controller specifically tailored to the pack acquires all sensor information like individual voltages and temperatures. Additionally, it processes and creates control signals for cell balancing. Approaches to decentralize the control of the battery pack are driven by the requirements of higher efficiency, modularity and easier integration in order to cope with the perennial demand for shorter design cycles and time-to-market.

Decentralization at the measurement level for reduced wiring and simplified matching has been proposed in several places. Stuart [169], for instance, proposes a pack consisting of 4 submodules where an individual ECU board controls 12 cells and communicates with a central unit. In [44], a similar master-slave setup is described that can additionally isolate individual cells for safety. A circuit with increased focus on efficiency and isolated communication has been presented more recently in [14]. [142] even uses wireless signals for communication to further reduce wiring and discusses the ensuing issues from electromagnetic interference. As all these approaches offload tasks from the master device to the computing devices of the individual cells, these tasks can also be scheduled in a way that creates more load on stronger cells, thus equalizing the pack and leading to more runtime [124].

Instead of communicating to a master for coordination as in the previous works, the smart cells introduced in [167] are fully autonomous, individually control their parameters, and coordinate their actions with other smart cells via communication.

Cell balancing techniques While cells that are connected in parallel balance according to their voltage, somewhat like capacitors, serially connected cells require additional circuitry. Generally, the available approaches are classified into *passive* and *active* cell balancing. Passive cell balancing dissipates excess energy of cells above minimum pack SoC via a resistor until they reach the charge level of the weakest cell [82]. While this technique is easy to implement and has low cost in comparison with the active cell balancing techniques, it suffers from low

efficiency since all excess energy is dissipated as heat across the balancing resistors. As cooling must bring this heat out of the system, balancing quickly is also challenging [170]. Commercial implementations use currents below $10\,\mathrm{mA}$, like [174] from Texas Instruments, or treat cells sequentially and with external resistors to reach $200\,\mathrm{mA}$, like [127] from Maxim Integrated.

Active approaches, by contrast, transfer energy from cells with high SoC to cells with low SoC instead of dissipating it. This significantly increases energy efficiency of the balancing process. In addition, transfers allow significantly higher currents as reduced dissipation also leads to less heat. Unlike the quasi-fixed design for passive balancing, there are many ways to achieve Active Cell Balancing (ACB). A qualitative overview of numerous techniques can be found in [102], [133], or [32]. These papers introduce the basic functionality of roughly a dozen techniques, focusing on component count, ease of implementation, and efficiency estimates. Many of these basic approaches are simulated in [52] although the evaluation of balancing time and efficiency remains qualitative in nature.

A systematic classification that encompasses many of the papers from the previous paragraphs has recently been presented in [69]. This publication structures ACB architectures by topology or equivalently by the kind of charge transfers that can be performed. The most important topologies are *Cell Bypass, Cell to Pack, Pack to Cell, and Cell to Cell*.

Cell Bypass is typically achieved via shunting as in [160] or [125]. Instead of transferring charge, these approaches reroute the external load current to temporarily reduce the stress on weak cells. Although they introduce some fault tolerance, or even reconfigurability [94], with moderate control effort, these techniques are often not an option because they insert shunts into the main connection of the battery. The so-created resistance entails an additional energy loss during normal operation which may be hard to overcome via balancing.

Unlike Cell Bypass, the other ACB topologies transfer charge. The available mechanisms to create such transfers are discussed in the next paragraph. Once a mechanism is selected, a suitable routing network can then form any of the following topologies, each with its own benefits and drawbacks.

Cell to Pack approaches transfer charge from one cell to the entire battery pack. In *Pack to Cell* techniques, conversely, the pack collectively sends to one cell. As such, both have very simple charge routing strategies. Cell to Pack is strictly preferable to Pack to Cell for two reasons [6]. First, the low voltage on the transmitting side can be handled by cheaper transistors. Second, the high output voltage leads to improved efficiency (80 % vs 70 %).

Cell to Cell approaches, the focus of this thesis, can be further grouped by whether or not they allow transfers between non-adjacent cells. In the literature, the focus is often on the simpler version, ignoring the more sophisticated circuits. According to [6], for instance, Cell to Cell leads to the most efficient one-to-one transfers (90 %). If charge is transferred in daisy chain fashion over several cells, however, this efficiency is greatly reduced. Although correct, this drawback is mostly overcome by non-adjacent transfers [88].

Quantitative guidelines for the performance of the various topologies are also obtained via optimization techniques. Please refer to Section 6.5 for a discussion on these works.

Other topology aspects like modularity and locally created actuation signals [138] are less discussed. The former helps when assembling a pack; the latter is necessary for distributed BMS because the alternative, clock synchronization in the microsecond range, is very difficult to achieve.

Charge transfer mechanisms While the routing network ultimately determines the topology, its design and the overall performance largely depend on the charge transfer mechanism, i.e., on the components that are used as temporary energy storage. The options include inductors, transformers, capacitors, and even combinations thereof.

Capacitors by themselves can be connected in parallel to transmitting and receiving cell in an alternating fashion. This requires a network of switches but actuation and timing remain simple. [143] implements adjacent Cell to Cell transfers in this way, [18] adds remote transfers. Their main drawback is that the parallel constellation inherently has poor efficiency. For this reason, capacitors are typically accessed via other components as in [15] or [121] where inductors are used. In such setups, the efficiency considerations of the auxiliary component by itself, as discussed in the following paragraphs, largely apply. Although an additional transfer becomes necessary since capacitors are not the ultimate destination, the isolating aspects of such approaches may lead to smaller switching networks.

Inductors rely on high-frequency switching signals (Section 5.2). These signals can become quite involved because inductors do not isolate the current meshes they are members of from one another. Although inconsequential for simple cases [103], this typically leads to high transistor counts for sophisticated topologies like the architectures with non-adjacent transfer from [88] or [123]. Once designed, however, inductor-based circuits operate very efficiently and require comparatively little space. Off the shelf, devices with PowerPump from Texas Instruments, like [173], enable transfers between neighboring cells with up to $1\,\mathrm{A}$ in this way. Please refer to [186] for a high-level description.

Transformers are an interesting alternative to inductors. Even though their actuation signals are relatively similar, the design is much simpler as transformers provide isolation and thus inherently prevent many short circuits. From an efficiency perspective, they are comparable to inductors. In simple setups, inductors have an advantage. They are smaller at equivalent parameters since transformers consist of two windings and they have less leakage terms. In larger setups, transformer architectures may require less transistors, however, and thus lead to improved cost and efficiency values. This is somewhat evident in circuits, like [162] or [80], that create a Cell to Cell topology with a switching network around a single transformer. It becomes even more clear for Cell to Pack variants. While these can be implemented with excellent efficiency (over $90\,\%$) with multi-winding transformers as in [113] or [60], they remain challenging for inductor architectures. Note, however, that it is not clear how to scale these approaches since it becomes difficult to create the required custom transformers for packs with more than 10 cells. Linear Technology offers such a balancer with Cell to Stack topology for up to 6 cells in series [116]. It is rated for currents up to $10\,\mathrm{A}$ if suitable transistors are added externally.

Functionality verification Charge transfer architectures are typically evaluated using circuit simulation or test hardware implementations. Since such manual techniques may not be sufficient in complicated cases, formal verification is also emerging in this domain. In [13], for instance, the access logic for a circular balancing bus is validated using SAL [54]. A more general approach that takes all electrical components into account, at least qualitatively, is pursued in [123]. There, the authors rely on a Boolean Satisfiability (SAT) solver to develop a GUI tool that formally verifies user-designed balancing architectures over all switching phases.

Li-Ion battery models Whether analyzing EV drive cycles or mobile phone runtime, a simulator needs battery models that can predict SoC, voltage characteristics, and dynamic behavior. For Li-Ion batteries, the families of *electrochemical, mathematical, and electrical models* compete to fill this role. Overviews of these options, as in [100] or [43], conclude with varying advice.

Electrochemical models offer the deepest insight into the battery [140]. For this reason, they are the only option when determining electrode size or other aspects of battery design. As they are a system of coupled time-varying Partial Differential Equations (PDEs), however, they are computationally expensive. In addition, many parameters they require are not provided by the manufacturer and "must be determined independently ... through a series of experiments ..." [55]. With these downsides in mind, it is nevertheless possible to simplify the complicated models to simpler, one-dimensional versions that are suitable for control applications and still have physically meaningful parameters [43].

Mathematical models predict battery behavior using empirical equations [153] or via a stochastic process [46]. While they can be evaluated quickly, their results may not reliable in practice. According to [45], such models lead to 5–20 % error in voltage characteristics and SoC evolution.

Electrical models are the most intuitive option for circuit simulation. Most of them are based on Thevenin's theorem, impedance networks, or runtime dependency [45]. A Thevenin-based model describes the cell behavior with a series resistor and one [192] or a number of stages [159] with resistor and capacitor connected in parallel. In this process, it is assumed the OCV is constant which prevents Thevenin models from tracking SoC evolution. Impedance-based models like [31] require complicated impedance spectroscopy to fit the parameters. Nevertheless, they are accurate only for a fixed SoC and cannot predict the runtime of a battery. Runtime-based models, on the other hand, employ a circuit network that simulates the runtime for a constant discharge current. As they are inaccurate for varying load currents, combined models become necessary. The model from [45], employed in this thesis, is such a model that provides accuracy for arbitrary load currents over the full SoC range using two resistor-capacitor stages. As there is a trade-off between accuracy and complexity when determining the number of these stages, other models have also emerged. Some demonstrate sufficient accuracy with only one resistor-capacitor stage, like [154] or [95]. Others argue for the need of a third stage [100].

Another aspect of battery modeling is aging. It is well known that parameters, like capacity or internal resistance, change over time. Since the corresponding experiments require months to complete, however, a consensus on the modeling of these effects has not been reached yet. While the main factors seem to be depth of discharge and mean SoC [157], the "degradation is more complicated than ... reported in the literature" [57] and larger studies show that significant randomness is involved [19].

State of Charge (SoC) estimation The charge within a battery cell, summarized in the SoC ratio, cannot be measured directly. Only terminal voltage and current are available to estimate the SoC. An accurate SoC is required for many applications, however, like the fuel gauge for an EV. It is also known that even passive balancing based on voltages finishes with significant offsets [186].

There are several approaches to find the SoC in a lab situation. Starting at a known value and

integrating the precisely measured input current, so-called Coulomb counting, is typically used as reference. Waiting for several hours and taking a voltage measurement may also reveal the OCV and lead to a reasonable estimate. There are other techniques from the chemistry domain, like impedance spectroscopy, but they are not suitable for estimation at runtime either.

A discussion on techniques that estimate SoC at runtime can be found in [151]. As the voltage profile of battery cells can be rather flat, a measurement accuracy of $5\,\mathrm{mV}$ for benign chemistries or even $1\,\mathrm{mV}$ for the most challenging cases is required to reliably estimate the SoC. The computational challenge is currently addressed mainly with *Kalman filters* and initial *adaptive observer* approaches.

Kalman filter approaches are comprehensively described in Plett's three-part overview [146–148]. Besides simpler mathematical models that consider SoC to be the only state, the author also investigates hysteresis and achieves estimation errors in the range of $2\,\%$. [110] uses an impedance-based battery model that they simplify to reduce computation time, aiming at sampling periods below $100\,\mathrm{ms}$ for multiple cells. This technique achieves SoC errors around $1.5\,\%$. In [74], by contrast, a Thevenin battery model is used. After parameter identification, the authors propose an adaptive component for the Kalman filter that reduces the mean estimation error to $1\,\%$

Adaptive observers A major difficulty for static battery models as employed by Kalman filters is the strong dependency on SoC that many parameters exhibit. According to [150], some of them vary as much as $800\,\%$. To address this, the authors adaptively estimate the model parameters along with the SoC at runtime. In this way, convergence times and estimation errors are significantly reduced. Another, less involved technique is described in [78] where the gain of a Luenberger observer is adapted at runtime to minimize the error along the convergence trajectory.

Electrochemical models are rarely used for SoC estimation in the literature. A noteworthy example operating an observer on a reduced version of the full kinetics can be found in [96]. While computationally expensive, it is not only highly accurate but also provides additional insight about other unmeasurable details like internal temperature.

A recent, thorough review of the various cell monitoring techniques can be found in [181].

6 Optimizing Efficiency in Active Cell Balancing (ACB)

Chapter 5 deals with modeling and quantitative evaluation of inductor-based ACB operation. This chapter deals with two subsequent elements in the overall design flow (see Sections 1.2 and 5.1): component optimization and routing strategies. First, it discusses a transfer model that is suitable for mathematical programming (Section 6.1). This model accepts some approximative transformations as they only lead to inaccuracy in very inefficient and hence insignificant operating points. With this model, we then examine a Geometric Programming (GP) formulation for inductor design (Section 6.2) and approaches that find the best transfer current (Section 6.3). The higher-level issue of charge routing is analyzed next, with the derivation of a bound on the achievable performance and the presentation of several strategies (Section 6.4). A discussion on related work (Section 6.5) concludes the chapter.

6.1 Optimization-friendly charge transfer model

In Section 5.8, several techniques for accurate, but rapid simulation of large-scale ACB scenarios have been discussed. While these models can be used to look for good operating parameters interactively, their nonlinearities are not suitable for efficient optimization algorithms. In order to systematically find the best settings even for large problems, this section relaxes accuracy requirements slightly to derive linear constraints that describe the transfer dynamics. In this form, ACB can be included in various mathematical programming frameworks and efficiently solved. Section 6.4.1, in particular, uses an LP solver based on this model after deriving suitable cost terms.

To derive the model, we assume the current interface from Fig. 5.13 on page 98 where a constant peak current is maintained by adjusting the timing settings as voltages evolve. A

simple battery model without RC stages is used as in Section 5.8.4. Please refer to (5.7) and the explanation there for a discussion on the errors this common simplification introduces. In this context, we examine the power series expansion of the closed-form terms for the intra-phase dynamics from Section 5.7 to reformulate the iteration from Section 5.8.2 as ODE.

Power series expansion of single-phase charge differences With the current interface from Fig. 5.13, the charge q_d of a single cycle can be calculated using (5.21). That formulation is reproduced here with the phase-dependent parameters of Table 5.2 from Page 97. It yields the charge differences q_t and q_r for transmitter and receiver, respectively.

$$\begin{aligned} q_t =& \frac{LV_t}{R_t^2}\log\Big(\frac{V_t - IR_t}{V_t}\Big) + \frac{L(I-0)}{R_t} \qquad = \frac{LV_t}{R_t^2}\log\Big(1 - \frac{IR_t}{V_t}\Big) + \frac{LI}{R_t} \\ q_r =& \frac{LV_r}{R_r^2}\log\Big(\frac{V_r}{V_r + IR_r}\Big) + \frac{L(0-(-I))}{R_r} = \frac{-LV_r}{R_r^2}\log\Big(1 + \frac{IR_r}{V_r}\Big) + \frac{LI}{R_r} \end{aligned} \tag{6.1}$$

Here, the relation $\log(\frac{1}{v}) = -\log(v)$ explains the second step for q_r. Now, consider, $\log(1+z) = z - \frac{z^2}{2} + \frac{z^3}{3} + \ldots$, a power series expansion of the natural logarithm which transforms q_t and q_r to the following, approximate representations.

$$q_t \approx \frac{LV_t}{R_t^2}\Big[\frac{-IR_t}{V_t} - \frac{1}{2}\Big(\frac{-IR_t}{V_t}\Big)^2 + \frac{1}{3}\Big(\frac{-IR_t}{V_t}\Big)^3\Big] + \frac{LI}{R_t} = -\frac{LI^2}{2V_t} - \frac{LI^3R_t}{3V_t^2} \tag{6.2}$$

$$q_r \approx \frac{-LV_r}{R_r^2}\Big[\frac{IR_r}{V_r} - \frac{1}{2}\Big(\frac{IR_r}{V_r}\Big)^2 + \frac{1}{3}\Big(\frac{IR_r}{V_r}\Big)^3\Big] + \frac{LI}{R_r} \quad = \frac{LI^2}{2V_r} - \frac{LI^3R_r}{3V_r^2} \tag{6.3}$$

As linearizing the phase duration (5.20) leads to $T_d = \frac{LI}{V}$, these terms can also be interpreted as follows.

$$q_t \approx -\frac{1}{2}IT_t - \frac{1}{3}\frac{I^2R_t}{V_t}T_t \qquad\qquad q_r \approx \frac{1}{2}IT_r - \frac{1}{3}\frac{I^2R_r}{V_r}T_r \tag{6.4}$$

Note, however, that T_t and T_r are not necessarily constant in these expressions!

ODE formulation On the way to the desired representation, the transfer dynamics are now considered in the form of a discrete ODE as in Section 5.8.3. In order to further simplify this ODE to forgo a numerical solver, only the first and major part of the expanded charge differences (6.2), (6.3) is included. Under this perspective, the voltage within one linear segment of model (5.8) from Section 5.4 evolves as follows.

$$\begin{aligned} V_t[k+1] =& V_{\zeta,i} + \zeta_i\big(Q_t[k+1] - Q_i\big) = V_{\zeta,i} + \zeta_i\big(Q_t[k] + q_t - Q_i\big) \\ \approx& V_{\zeta,i} + \zeta_i\big(Q_t[k] - \frac{LI^2}{2V_t[k]} - Q_i\big) = V_t[k] - \zeta_i\frac{LI^2}{2V_t[k]} \end{aligned} \tag{6.5}$$

Here, k refers to discrete switching cycles. As the cycles are so numerous in ACB, this recurrence relation can also be considered as ODE $\frac{\Delta V_t}{\Delta k} = -\zeta_i\frac{LI^2}{2V_t}$. This ODE for V_t and the similar version for V_r have the following unique positive solutions.

$$V_t[k] = \sqrt{V_t^2[0] - \zeta_t LI^2 k} \qquad\qquad V_r[k] = \sqrt{V_r^2[0] + \zeta_r LI^2 k}. \tag{6.6}$$

Clearly, a transformation via series truncation is not lossless. The resulting error consists of the higher-order terms in (6.2), (6.3) that are omitted. The first and largest term are the dissipative losses as explained in the next paragraph. Like all terms, it grows with IR/V. The model is thus most accurate in the efficient operating areas where R, I and consequently losses are small. More details on this error that other optimization models share can be found later in this section (Fig. 6.1 and Table 6.1).

Dissipative losses The second terms from (6.2), (6.3), which have been omitted so far, represent the dissipative or transfer loss E_{tf}. This is somewhat evident from (6.4). It can also be calculated more concretely by examining the energy balance of the two participating cells. For this purpose, we continue assuming that the cell voltage remains constant during individual switching cycles as in Section 5.7.

$$\begin{aligned} E_{tf} =& \Delta E_t + \Delta E_r = V_t q_t + V_r q_r \\ =& V_t\Big[-\frac{LI^2}{2V_t} - \frac{LI^3 R_t}{3V_t^2}\Big] + V_r\Big[\frac{LI^2}{2V_r} - \frac{LI^3 R_r}{3V_r^2}\Big] = -\frac{LI^3 R_t}{3V_t} - \frac{LI^3 R_r}{3V_r} \end{aligned} \tag{6.7}$$

Energy-based reformulation Although the voltage evolution from (6.6) can be used for fast simulation, it is still unwieldy in optimization scenarios. We can, however, reformulate it to be based on energy instead of voltage. This renders the main dynamics linear with respect to cycles k given constant peak current I. If the current is calculated by the platform, as described in Section 5.5.3, this can be used to solve large-scale routing problems (Section 6.4.1).

Switching the state variable from voltage to energy is straightforward. We substitute (6.6) into battery energy (5.12) and obtain the following for the transmitting cell.

$$E_t[k] = E_{\zeta,i} + \frac{1}{2\zeta_i}\big[\sqrt{V_t^2[0] - \zeta_t LI^2 k}^{\,2} - V_{\zeta,i}^2)\big] = E_{\zeta,i} + \frac{1}{2\zeta_i}\big[V_t^2[0] - V_{\zeta,i}^2\big] - \frac{1}{2}LI^2 k$$

Transforming the voltage terms back to $E_t[0]$ and applying similar algebraic transformations to the receiver side, leads to the desired transfer dynamics:

$$E_t[k] = E_t[0] - \frac{1}{2}LI^2 k =: E_t[0] - \Delta E \qquad\qquad E_r[k] = E_r[0] + \Delta E \tag{6.8}$$

Tracking time evolution The transfer dynamics (6.8) from this section are calculated in terms of switching cycles. Similarly to Section 5.8.3, the evolution of time must thus be tracked separately. For this purpose, consider the following power series expansion for T_d from (5.20).

$$T_d \approx \frac{LI}{V} + \operatorname{sgn}(I)\frac{LI^2 R}{2V^2} + \frac{LI^3 R^2}{3V^3} \tag{6.9}$$

This formula calculates the time for one switching cycle by considering $T_c = T_t + T_r$. Depending on the optimization problem, it can be included as constraint with varying number of terms. For simulation, we discuss a more accurate option at the end of this section.

Why a better model with voltage dependency? Optimization for ACB is often performed with the standard linear state space representation

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{6.10}$$

Since the cells in ACB barely evolve without actuation, the system matrix can be selected as $A = 0$. Although it is not always clear how the exact dynamics from Section 5.3 are then mapped to this representation, the most common approach is to introduce an average current that can be actuated directly.

$$\dot{Q}(t) = \beta i(t) \tag{6.11}$$

Looking at the first terms from (6.4), it may appear that $\beta = \frac{1}{2}\frac{T_t}{T_t+T_r}$ would be a reasonable choice for the transmitting cell. As β must be constant, however, and T_t, T_r depend on cell voltages, among other things, this choice can only be accurate in a certain working point. This is problematic because ACB deliberately moves the system to a new state unlike typical control applications that maintain a working point.

Outside its working point, an average current model quickly leads to relative errors in the range of $5\,\%$, as shown in Fig. 6.1. This figure shows an experiment with two cells having linear voltage behavior for 3.0–$4.5\,\mathrm{V}$. Resistances and inductance were modeled as $R_t = R_r = 10\,\mathrm{m\Omega}$ and $L = 100\,\mu\mathrm{H}$, respectively. The SoC of the receiving cell was set to $z_r = 0.5$ while the transmitting cell was varied. Brief transfers of $\Delta t = 0.1\,\mathrm{s}$ with a peak current of $I = 1.25\,\mathrm{A}$ are simulated using three techniques: the iterative approach from Section 5.8.2 as reference, the energy-based technique (6.8) proposed here, and the average current model (6.11). The relative errors $\epsilon_{rel} = (\Delta z_{ref} - \Delta z)/\Delta z_{ref}$ with respect to the reference are plotted in Fig. 6.1.

In spite of the growing inaccuracy, the average current approach can still be justified for SoC differences $|\Delta z| < 0.04$ that lead to $\epsilon_{rel} < 1\,\%$. Even though this includes a significant part of the relevant ACB scenarios, the energy-based model (i) is strictly superior with higher accuracy and neither higher computation costs nor more complicated optimization. In addition, it (ii) must not fear future cells that may introduce steeper voltage profiles.

It is worth pointing out that both approximative models intersect at $\Delta z = 0$ where they are mathematically equivalent since β from (6.11) is calibrated. This offset depends on resistance as well as current; it roughly corresponds to the dissipative loss terms identified in (6.7) that have been omitted so far. Please refer to Table 6.1 for more details.

Improving accuracy in simulation There are several possibilities to include the dissipation terms that create the offset in Fig. 6.1. For optimization, they are compatible with the Geometric Programming (GP) framework in some cases (Section 6.2). In other cases, it is necessary to account for them more coarsely (Section 6.4.1). For simulation, however, these terms can be included a posteriori for high accuracy.

With the voltage evolution as per (6.6), the dissipative losses from (6.7) can be integrated

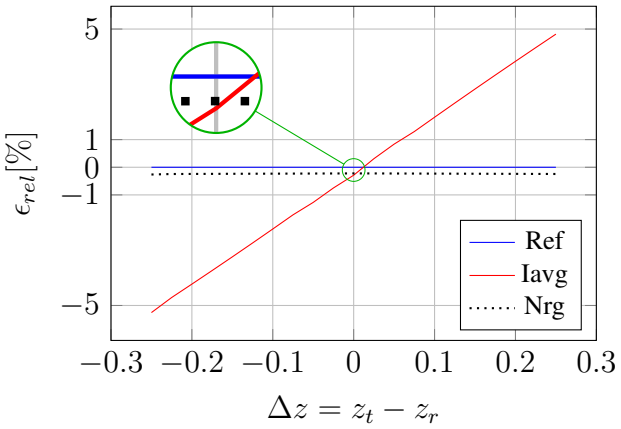


Figure 6.1: *The energy-based model (Nrg) from this section omits dissipation terms that lead to a small error if not dealt with using one of the later-suggested methods. In addition to these same terms (note intersection at working point $\Delta z = 0$), a typical state space representation (Iavg) also ignores voltage-dependency. This leads to relative errors of up to $5\,\%$ away from the working point.*

over time to obtain an accurate total.

$$\begin{aligned}\tilde{E}_{tf,t}[K] &= \int_0^K \frac{LI^3R_t}{3V_t[k]}\,\mathrm{d}k = \frac{2IR_t}{3\zeta_t}\big(V_t[0] - V_t[K]\big)\\ \tilde{E}_{tf,r}[K] &= \int_0^K \frac{LI^3R_r}{3V_r[k]}\,\mathrm{d}k = \frac{2IR_r}{3\zeta_r}\big(V_r[K] - V_r[0]\big)\end{aligned} \tag{6.12}$$

These transformations have been found using a Computer Algebra System (CAS) [187]. Note that current I reappears in the ending voltage $V_t[K]$ here. This makes these formulations difficult to use for optimization purposes. Subtracting them a posteriori in simulation does increase accuracy, however. Similar correction terms for the switching loss terms from (5.13) can be found in an analog fashion. Set $C_{oss} = 0$ for the alternative representation from (5.15).

$$\begin{aligned}\tilde{E}_{sw,t} =& \frac{C_{oss}K}{2}\big(V_t^2[0] - \frac{\zeta_t}{2}I^2KL\big) + \frac{1}{3}\tau_d IKV_t[K] + \frac{\tau_d V_t^2[0]}{3IL\zeta_t}\big(V_t[0] - V_t[K]\big)\\ \tilde{E}_{sw,r} =& \frac{C_{oss}K}{2}\big(V_r^2[0] + \frac{\zeta_r}{2}I^2KL\big) + \frac{1}{3}\tau_u IKV_r[K] + \frac{\tau_u V_r^2[0]}{3IL\zeta_r}\big(V_r[K] - V_r[0]\big)\end{aligned} \tag{6.13}$$

The balancing duration can also be derived more accurately by substituting the voltage evolution from (6.6) into (6.9) and calculating $\int_0^K T_t + T_r$ with a CAS. For both sides, this uses

$$\int_0^K T_d \approx \mathrm{sgn}(I)\left\{2\frac{V[0]-V[K]}{I\zeta} - \frac{R}{2\zeta}\log\left[1 - \mathrm{sgn}(I)\frac{I^2KL\zeta}{V[0]}\right] + \frac{2IR}{3\zeta}\left[\frac{1}{V[K]} - \frac{1}{V[0]}\right]\right\}. \tag{6.14}$$

These adjustments significantly improve the accuracy, as demonstrated by the experiment from Table 6.1. There, transfers with peak current $I = 1.0\,\mathrm{A}$ and a duration of $\Delta t = 0.5\,\mathrm{ms}$ are performed between two cells with voltages $V \approx 3.65\,\mathrm{V}$. The transfers use a variety of inductances and resistances and are evaluated with three iterative mathematical models. (6.2), (6.3)

furnish the single-term model, as for long-term model (6.6), and the two-term model to represent the corrections from the previous paragraphs. The full accuracy model from (6.1) serves as reference. Inductance and associated frequency barely affect accuracy in this experiment; further investigation is needed to confirm this for other approximations like timing or losses. The error does increase with resistance as expected, on the other hand; the resistance of circuits for battery cells that balance at $I = 1.0\,\mathrm{A}$ should hence be discussed. Assuming SAMSUNG cells [156], as in Section 5.4, with a capacity of $2.5\,\mathrm{A\,h}$, balancing with a peak current of $\frac{1}{4}C$ would correspond to two parallel cells and hence $R_C = 10\,\mathrm{m\Omega}$ cell resistance. As other circuit components typically contribute resistance in the same range, such a circuit can be analyzed with relative errors under $1\,\%$ even with the basic version. Larger battery backs that require higher currents also have lower resistances. While this truncation error is larger than the errors in the simulation models from Section 5.8, the model from this section improves upon alternatives and applies to a wide range of scenarios. Special care is only required for very fast and wasteful balancing.

Table 6.1: *Over various inductance (vertical) and resistance (horizontal) values, both basic and corrected optimization model remain sufficiently accurate for most scenarios (below $1\,\%$ relative error). Only inefficient conditions, with high resistance and current, that an optimization solver would avoid, lead to significant inaccuracies.*

	Basic (single term)			**Corrected (two terms)**		
Rel. Error	0.005	0.05	0.5	0.005	0.05	0.5
$3.0e{-6}$	$9.2e{-4}$	$9.3e{-3}$	$1.1e{-1}$	$9.4e{-7}$	$9.6e{-5}$	$1.2e{-2}$
$1.0e{-5}$	$9.2e{-4}$	$9.3e{-3}$	$1.1e{-1}$	$9.4e{-7}$	$9.6e{-5}$	$1.2e{-2}$
$3.0e{-5}$	$9.2e{-4}$	$9.3e{-3}$	$1.1e{-1}$	$9.4e{-7}$	$9.6e{-5}$	$1.2e{-2}$
$1.0e{-4}$	$9.2e{-4}$	$9.3e{-3}$	$1.1e{-1}$	$9.4e{-7}$	$9.6e{-5}$	$1.2e{-2}$

6.2 Inductor optimization via Geometric Programming (GP)

During the design phase of an ACB architecture, the individual components are selected once the circuit layout is determined. In this context, we now look for the inductor that performs best under several predetermined scenarios. To that end, we take the model from Section 6.1 and adapt it for a GP formulation. This formulation is then extended by inductor constraints and yields designs that are superior to off-the-shelf options in a case study.

Since the inductor is optimized by itself in this section, its resistance R_L must be treated individually and not as part of the aggregated resistances described in Section 5.3. We hence introduce $R_{t,0}$, $R_{r,0}$ to account for all other resistances in the equivalent circuit.

$$R_t = R_{t,0} + R_L \qquad R_r = R_{r,0} + R_L \tag{6.15}$$

Geometric Programming (GP) Instead of using a nonlinear solver, this section operates in the GP paradigm (i) to guarantee that an optimal solution is found and (ii) to deal with larger

problems. According to Boyd's overview [25], a GP problem in standard form is:

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq 1 \quad \forall i = 1, \ldots, m \\ & g_i(x) = 1 \quad \forall i = 1, \ldots, p \end{array}$$

where $x = \left[x_1, \ldots, x_n\right]$ is a vector of positive real-valued decision variables. f_i and g_i are posynomial and monomial functions, respectively. A monomial has the following form:

$$m(x) = c x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n} \qquad c > 0, a_i \in \mathbb{R}$$

Posynomials are sums of monomials and closed under addition, multiplication, positive scaling as well as division by monomials. Geometric Programming (GP) is a special form of convex optimization. GPs have polynomial time computational complexity and can be solved very efficiently by a variety of off-the-shelf solvers.

Adjusting charge transfer and losses for GP Instead of dealing with the actual losses, the technique in this section is content to minimize an upper bound. For this purpose, it considers only the highest and lowest voltages which the two cells of a certain link can reach. As cell voltage increases monotonously with SoC, the initial voltages of transmitting and receiving cell, $V_t[0]$ and $V_r[0]$, represent such bounds.

Inserting these bounds into the approximate charge differences from (6.2), (6.3), we obtain

$$q_{t,\max} \approx -\frac{LI^2}{2V_r[0]} - \frac{LI^3 R_t}{3V_r^2[0]} \qquad q_{r,\min} \approx \frac{LI^2}{2V_t[0]} - \frac{LI^3 R_r}{3V_t^2[0]}. \tag{6.16}$$

Minimizing transfer losses E_{tf} from (6.7) can be done directly. This formulation is a posynomial in R_t, R_r, I, L, V_r, V_t. After inserting bounds $V_t[0]$ and $V_r[0]$, it becomes the following.

$$|E_{tf}| = \frac{LI^3 R_t}{3V_r[0]} + \frac{LI^3 R_r}{3V_t[0]} \tag{6.17}$$

The switching losses, as described in Section 5.6, are also a posynomial in I, V_r, V_t. With $V_t[0]$ as upper bound for the voltage, (5.13) becomes

$$E_{sw} = \frac{1}{2}(\tau_d + \tau_u) I V_t[0] + C_{OSS} V_t^2[0] \tag{6.18}$$

Inductor Design Constraint In order to minimize both transfer and switching losses, we would like to have an inductor with microscopic resistance and large inductance that supports gigantic peak currents. However, there is obviously a trade-off involved between these features. Using the procedure described in Chapter 14 "Inductor Design" of [63], we begin by ensuring that the following inequality holds.

$$K_g \geq \frac{\rho L^2 I_{\text{max}}^2}{B_{\text{max}}^2 R_L K_u} \tag{6.19}$$

Here, K_g is the core geometrical constant summarizing the size and the layout of the core. It can be obtained from data tables. $\rho = 1.724\text{e-}6\,\Omega\text{cm}^{-2}$ is the resistivity of copper wire. $I_{\max}$ can be assumed to be the peak current I for non-dominated designs. K_u is the winding fill factor and is in the range of $K_u = 0.7$. Finally, we set $B_{\max} = 0.3\,\text{T}$, as recommended for most ferrite core materials in [98] (Chapter 9.4.5 "Design of Inductors").

Given the dimension of the overall circuitry, we select a suitable core which determines the geometrical constant K_g. From any combination of L, R, I that fulfills (6.19), we can then go back to the procedure from [63] and calculate air gap length l_g as well as number of turns n. This means that selecting L, R, I *implicitly fixes the entire inductor design*.

Timing considerations When equalizing a charge difference ΔQ between two cells, the number of switching cycles, K, is bounded below by $K \geq \Delta Q/(q_{r,\min} - q_{t,\min})$. This can be reformulated to

$$\frac{\Delta Q}{K} \leq \frac{LI^2}{2V_r[0]} + \frac{LI^3 R_t}{3V_r^2[0]} + \frac{LI^2}{2V_r[0]} - \frac{LI^3 R_r}{3V_r^2[0]} = \frac{LI^2}{V_r[0]} + \frac{LI^3\big(R_{t,0} + R_L - R_{r,0} - R_L\big)}{3V_r^2[0]}.$$

Here, inductor resistance R_L has been separated from R_t, R_r as described in (6.15). Rearranged, this yields a posynomial formulation:

$$\frac{\Delta Q}{K} + \max\left\{0, \frac{LI^3(R_{r,0} - R_{t,0})}{3V_r^2[0]}\right\} \leq \frac{LI^2}{V_r[0]} \tag{6.20}$$

Here, the $\max$ operation can and must be evaluated beforehand as the remaining circuit resistances $R_{t,0}$, $R_{r,0}$, are constant from the perspective of this optimization.

With the total cycles K thus determined, a bound on the total balancing time can be obtained by adapting (6.9). The following constraint ensures that this duration remains below a threshold $T_{\max}$.

$$T_t + T_r \leq \frac{LI}{V_r[0]} + \frac{LI}{V_r[0]} + \frac{LI^2 R_t}{2V_t^2[0]} + \frac{LI^3 R_t^2}{V_r^3[0]} \leq \frac{T_{\max}}{K} \tag{6.21}$$

In this formulation, only the second inequality is part of the optimization. The higher-order terms from T_r cannot be included because the second term is negative and thus not a posynomial.

Overall GP formulation Together, the posynomial terms from the previous paragraphs yield the following optimization problem that fits the GP paradigm.

$$\begin{aligned} \min \quad & \sum_{s\in\mathcal{S}} K^s \cdot (E_{tf}^s + E_{sw}^s) \quad \text{using (6.17), (6.18)} \\ \text{s.t.} \quad & \text{Inductor constraint (6.19)} \quad \forall I^s \\ & \text{Cycle count constraint (6.20)} \quad \forall K^s \\ & \text{Time constraint (6.21)} \quad \forall T_{\max}^s, K^s \end{aligned} \tag{6.22}$$

Problem (6.22) is solved using CVX, a MATLAB package that suitably reformulates geometric programs to equivalent convex representations [73].

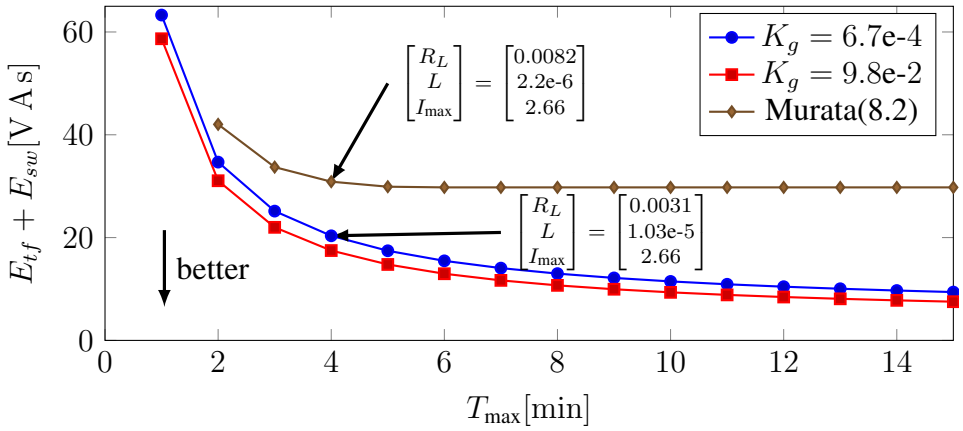


Figure 6.2: *Specifically designing the inductor reduces the average energy losses for the scenarios in Table 6.2 of a fixed inductor at least by 20%.*

Design quality For a demonstration of how useful this optimization is in practice, consider a design with off-the-shelf components that was deemed optimal by the search algorithm from [138]. This design uses the MURATA(8.2) inductor with effective volume $V_e = 893\,\mathrm{mm}^3$ and the ONSEMI(7.8) MOSFET. This entails the following circuit parameters.

$$\begin{bmatrix} R_{t,0} & R_{r,0} & \star \\ C_{OSS} & \tau_d & \tau_u \end{bmatrix} = \begin{bmatrix} 35\,\mathrm{m\Omega} & 35\,\mathrm{m\Omega} & \star \\ 125\,\mathrm{pF} & 44\,\mathrm{ns} & 168\,\mathrm{ns} \end{bmatrix} \tag{6.23}$$

The optimization designs are based on cores POT1107 and ETD29, as described, e.g., by the data tables in Appendix D of [63]. POT1107 has geometric constant $K_g = 6.67\text{e-}4$ and effective volume $V_e = 251\,\mathrm{mm}^3$ and should lead to a final product with volume in the range of the MURATA(8.2). The larger ETD29 with $K_g = 9.78\text{e-}2$ and $V_e = 5470\,\mathrm{mm}^3$ is included for comparison. Volume data has been obtained from the corresponding part numbers 0R42929EC and 0F41107UG in the Magnetics catalog.

The scenarios we take into consideration are chosen such that both cells remain within a typical SoC range of $[10\,\%, 90\,\%]$ with a maximum difference of 10 percentage points. They are detailed in Table 6.2.

Fig. 6.2 shows the results from this experiment. Each of the marks represents an inductor design that is optimal for a certain allotted balancing time $T_{\max}$ and core with geometrical constant K_g. All the curves flatten out once $T_{\max}$ is so large that time constraint (6.21) is not active in the optimization any longer. For $T_{\max} = 1\,\mathrm{min}$, the required current is too large for the fixed

Table 6.2: *Scenarios for inductor design, specifying SoC z, voltage V, and charge difference for transmitter and receiver*

Scenario	s_1	s_2	s_3	s_4	s_5
$z_t \vert V_t$	$0.9 \vert 3.315\,\mathrm{V}$	$0.76 \vert 3.284\,\mathrm{V}$	$0.55 \vert 3.273\,\mathrm{V}$	$0.41 \vert 3.262\,\mathrm{V}$	$0.25 \vert 3.220\,\mathrm{V}$
z_r, V_r	$0.82 \vert 3.304\,\mathrm{V}$	$0.72 \vert 3.279\,\mathrm{V}$	$0.49 \vert 3.271\,\mathrm{V}$	$0.33 \vert 3.240\,\mathrm{V}$	$0.18 \vert 3.197\,\mathrm{V}$
ΔQ[C]	316.5	172.8	189.2	300.2	287.8

inductor. Otherwise, the comparable designs from the proposed approach dissipate at least $20\,\%$ less energy ($\frac{42\,\mathrm{V\,A\,s}}{34.7\,\mathrm{V\,A\,s}} = 1.21$ for $T_{\max} = 2\,\mathrm{min}$). The larger core yields only mediocre further benefits.

Furthermore, the effects from using bounds for voltages and the power series reformulations (6.2) and (6.3) were evaluated. The relative errors that arise from these effects remain well under $1\,\%$ in the optimal points of GP problem (6.22).

Scalability To measure the performance of GP instance (6.22) we select the scenarios randomly by drawing voltage pairs according to $V \sim \mathcal{N}(3.15, 0.1)$, using the larger one as sender and calculating ΔQ according to the OCV mapping. We solve GP instance (6.22) with increasing $|\mathcal{S}|$ and record the required runtime. This procedure yields Table 6.3. Clearly, the scalability is excellent – almost linear – probably because the GP formulation is quite sparse. Even larger problems are solved in merely minutes on the computer we utilized (Intel i5-2540M @ $2.60\,\mathrm{GHz}$ with $8\,\mathrm{GB}$ RAM).

Table 6.3: *Runtime measurements for growing scenario vector*

# Scenarios $\lvert\mathcal{S}\rvert$	1	5	10	50	100
Runtime [s]	2.45	9.33	15.36	82.14	179.97

6.3 Optimal current for individual links

In this section, we discuss how to calculate an optimal operating current locally, i.e., for a single link (Definition 5.2). The ΔE-interface (Fig. 5.15 on page 99) needs such methods to drive the system in operation. Besides more efficient operation, locally calculating the current also yields a priori loss bounds for the best case reference solution (Section 6.4.1).

An initial idea may be to calculate the current that minimizes losses per time. This typically leads to very small currents, however, that do not create significant progress. For this reason, we instead consider ℓ, relating losses from a single cycle to the amount of energy moved at the same time.

$$\ell(V, \mathbf{I}) := \frac{E_{tf}(V, \mathbf{I}) + E_{sw}(V, \mathbf{I})}{\Delta E} \tag{6.24}$$

Minimizing this formulation ensures that any surplus is transferred as efficiently as possible. This can be done with a restricted version of the GP formulation from Section 6.2. As GP solvers may not be available at runtime, however, this might require a lookup table.

For a single link, we can also minimize (6.24) directly. We first present the calculation of the optimal current in closed form for environments where using the simpler switching loss expression (5.15) is justified. We then comment on the more conservative, complicated case (5.13). We forgo the time constraint at first and form the derivative of the losses to transferred energy

ratio. Here, we utilize $\Delta E = \frac{1}{2}LI^2$ as given in (6.8) and the dissipative losses from (6.7) in addition to the switching losses from (5.15), as discussed.

$$\begin{aligned}\ell_t(V_t, I) + \ell_r(V_r, I) =& \frac{E_{tf,t} + E_{sw,t} + E_{tf,r} + E_{sw,r}}{\Delta E} \\ =& \frac{2}{LI^2}\Big[\frac{LI^3R_t}{3V_t} + \frac{LI^3R_r}{3V_r} + \frac{\tau_t I V_t}{2} + \frac{\tau_r I V_r}{2}\Big]\end{aligned} \tag{6.25}$$

We now derive with respect to I and look for a root of this term to find an optimal value.

$$\frac{2R_t}{3V_t} + \frac{2R_r}{3V_r} - \frac{\tau_t V_t}{LI^2} - \frac{\tau_r V_r}{LI^2} = 0 \quad \Leftrightarrow \quad I^2\Big(\frac{2R_t}{3V_t} + \frac{2R_r}{3V_r}\Big) = \frac{\tau_t V_t}{L} + \frac{\tau_r V_r}{L} \tag{6.26}$$

As we are only interested in positive currents, the optimal current is given by

$$I_{\text{opt}} = \sqrt{\frac{3}{2L}\frac{\tau_t V_t + \tau_r V_r}{\frac{R_t}{V_t} + \frac{R_r}{V_r}}}. \tag{6.27}$$

If you prefer using the more conservative terms from Eq. (5.13) for the switching losses, the approach we just followed (derivation and equation solving) requires significantly more effort. In particular, $\frac{\mathrm{d}}{\mathrm{d}I}\frac{E_{tf}+E_{sw}}{\Delta E} = 0$ is now a cubic equation. A CAS [187] still yields a closed-form solution for this equation that can be readily implemented, just not easily printed. Alternatively, cubic equations can also be solved efficiently at runtime.

Now recall that optimal current I_{opt} from (6.27) is calculated without taking a time constraint into account. As long as I_{opt} fulfills the speed requirement, this is no issue and the optimal current can directly be utilized. If I_{opt} is too slow however, we can select the smallest current that fulfills the time constraint. The convexity of the cost function guarantees that this is the best choice if we require more speed.

A starting point for the time calculation is the total time from (6.9). With four terms, two for T_t and two for T_r, inverting (6.9) leads to a quartic equation. Although efficiently solvable, relaxing accuracy and solving only for two terms in closed form may be a better option. Alternatively, finding a bound with fixed voltages as in Section 6.2 is also viable.

6.4 Charge routing problem

The previous sections have dealt with the analysis and optimization of single links in ACB architectures. A link is specified by transmitter and receiver cell that can transfer cells in some way. Please refer to Definition 5.2 for more details.

This section takes a higher-level perspective and deals with routing strategies. These strategies, in the scope of this thesis, build on top of the interfaces from Chapter 5. They decide which link should be actuated and how.

Fig. 6.3 depicts how a routing strategy interacts with the balancing platform. The platform consists of cells and links that are created by balancing components, like inductors. It provides information about the state of the cells, namely their voltage and their charge or SoC.

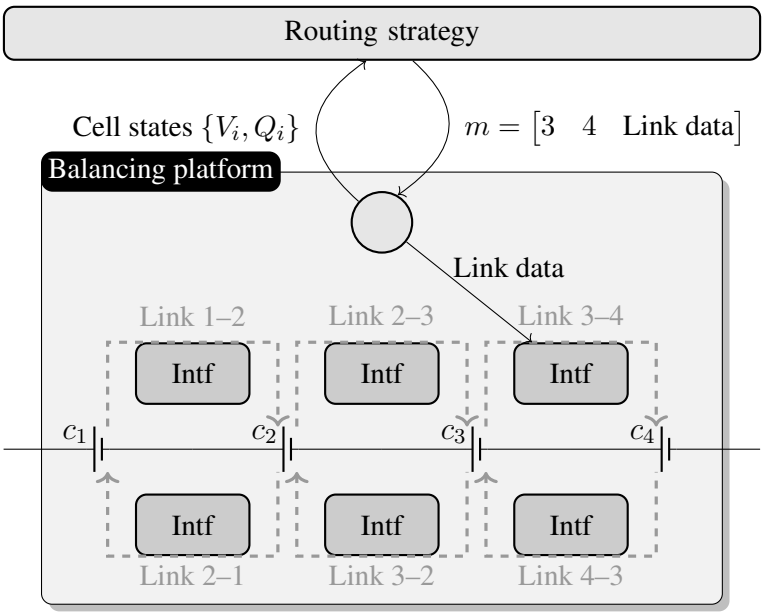


Figure 6.3: *The routing strategy receives the current cell states from the balancing platform. It actuates the platform by sending* movements*, as specified in Section 5.5. Here a movement is sent for link 3–4. The link data is forwarded to the specified link where it is further processed according to the corresponding interface (intf) before the cells are actuated physically.*

Using this information, the routing strategy forms movements, as defined in Section 5.5 (Definitions 5.4, 5.5, and 5.6). The link data from such a movement, e.g., $\begin{bmatrix} I & \Delta t \end{bmatrix}$ for an I-movement, is forwarded to the respective link. There, it is processed by the corresponding actuation interface to actuate the balancing hardware physically.

Before we discuss approaches for charge routing, we now clarify what the problem actually looks like. We are given a number of cells with associated links. In this context, cells are taken into account according to the following definition.

Definition 6.1 (Cell)**.** *A cell is characterized by Q_{max}, its capacity, and Q, the charge it currently holds. The cell interacts with other parts of the system via terminal voltage V and current i. This is summarized by the state space representation*

$$\dot{Q}(t) = -\, i(t) \qquad\qquad V(t) = f\big(Q(t), i(t)\big).$$

Options for f, the mapping between charge and voltage, are discussed in Section 5.4.

An additional complication comes from the fact that not all links can be operated simultaneously. For instance, we can typically charge one cell only from a single other cell at any given time. The following definition formalizes this concept.

Definition 6.2 (Link junction)**.** *A link junction is a segment shared by multiple links. It is described by a set $\mathcal{J} = \{l_i\}_i$ containing the links that utilize it. Only one link can transfer over any segment at a time, leading to constraints of the form*

$$\sum_{i \in \mathcal{J}} \nu_i \leq 1 \tag{6.28}$$

where $\nu_i \in \{0, 1\}$ indicates whether a link is currently active or not.

Junctions are tough to model efficiently. In their original form, they lead to an integer problem. Integers are inherently challenging in optimization and even harder to combine with nonlinear effects. For this reason, it is often preferable to model junctions in other ways.

One alternative is to consider $\nu_i \in [0, 1]$ as continuous variables. In this way, one models the utilization of the corresponding segment and ensures that it remains below $100\,\%$ over a longer period of time. Since there are so many discrete steps, a round-robin scheduling can approximately create virtually all ratios that an optimization may come up with. Consider also the related argument in Section 5.8.3 for reference.

Another alternative, pursued here, exclusively looks at the time that a segment is used. Added up, that time must be smaller than the total time of the system. Here, system time may refer to a constraint on the balancing time or the length of a time step in discrete-time formulations. Note that this is equivalent to the aforementioned continuous utilization variables.

Given cells with their states, links, and junctions, the *basic balancing problem* consists of finding a state where deviation from the mean $\|Q_c - \bar{Q}\| = 0$ in a *specified time frame* $T_{\max}$ while *dissipating as little charge as possible*. Equivalently, we maximize the total end charge.

$$\max \quad \sum_{c=1}^{N} \mathbf{Q}_c[t_K] \tag{6.29}$$

$$\text{s.t.} \quad \mathbf{Q}_c[t_{k+1}] = \mathbf{Q}_c[t_k] + \sum_{l \in \text{IN}(c)} q_{r(l)}(\mathbf{Q}_{r(l)}[t_k], \mathbf{I}_l, \Delta\mathbf{t}_{l,k}) - \sum_{l \in \text{OUT}(c)} q_{t(l)}(\mathbf{Q}_{t(l)}[t_k], \mathbf{I}_l, \Delta\mathbf{t}_{l,k}) \quad \forall c = 1, \ldots, N \tag{6.30}$$

$$t_{k+1} \geq t_k + \sum_{l \in \mathcal{J}_i} \Delta\mathbf{t}_{l,k} \quad \forall \mathcal{J}_i \in \{\mathcal{J}_i\}_i, 0 \leq k \leq K-1 \tag{6.31}$$

$$\mathbf{Q}_c[t_K] = \mathbf{Q}_{c+1}[t_K] \quad \forall c = 1, \ldots, N \tag{6.32}$$

$$\Delta\mathbf{t}_{l,k} \geq 0 \quad \forall l \in \mathcal{L}, 0 \leq k \leq K-1$$

$$t_K \leq T_{\max}$$

We are showing variables in optimization programs in bold for clarity. K, the number of required time steps, is unknown a priori but only has to be chosen sufficiently large in practice. The optimization can decide not to use superfluous time steps by letting $\Delta t_{l,k} = 0$ there.

(6.30) represents the transfer dynamics. q_t, q_r can be implemented by any of the simulation models from Section 5.8. In this case, (6.30) becomes a non-convex constraint. This makes the problem very hard to solve. The energy-based model from Section 6.1 which leads to linear constraints is thus often preferred. (6.31) ensures that all junctions are used only as long as the length of a time step allows, as discussed previously. (6.32), on the other hand, ensures that we end up with a battery pack where all cells are balanced.

The size of the problem depends mostly upon how many links connect the cells to each other. The roughly 100 serially connected cells of contemporary EVs lead to a problem of at least 500 variables. If remote transfers are allowed and the fully connected graph of the cells is considered, more than 10000 variables are required. These numbers assume that all transfers

occur in a single time step ($K = 1$) which is, of course, not accurate enough in many cases. LP solvers easily handle problems of this size. For nonlinear solvers that we require in the current form, problems of 500 or 10000 variables are considered large and extremely large, respectively.

In the following, we analyze the charge routing problem quantitatively. As the optimal current can be calculated for each link individually (Section 6.3), we can find a best case estimate for the transfer costs. With this bound, it becomes possible to calculate the best case for overall charge routing in a Linear Programming (LP) formulation (Section 6.4.1). This reference value demonstrates that heuristic strategies already yield good performance in non-concurrent scenarios. For this reason, we subsequently present a strategy that focuses on simplicity, embedded implementation, and parallelization (Section 6.4.2). The section ends with an experimental comparison of this strategy, a heuristic strategy, and the reference LP (Section 6.4.3).

6.4.1 Best case reference solution for charge routing

With the more manageable model for the transfer dynamics from Section 6.1 and locally calculated current as per Section 6.3, it is possible to find a lower bound for the losses in a certain balancing scenario. Such a bound, representing an unachievable best case, is highly valuable to evaluate routing strategies and gauge the potential for improvement.

Consider how the optimal current is calculated for individual links (Definition 5.2) in Section 6.3. Any solution, like (6.27), depends on cell voltages and link parameters, with or without additional time constraint. As the involved link parameters are constant, the optimal current varies only with the evolving cell voltages. While the best case for the transfer losses occurs at V_{max} the switching losses are lowest at V_{min}, e.g., in (6.25). If we substitute these best cases – even though they cannot occur simultaneously – we obtain the best case losses ℓ_{min}. ℓ_{max} is calculated analogously. This works with any of the approaches described in Section 6.3. We do this for each of the links (Definition 5.2) in our network and arrive at a best case bound for the losses that we must incur for each block of energy that we transfer across. Given these optimistic link prices, we then formulate the overall best case problem.

$$
\begin{aligned}
\min \quad & \sum_{l \in \mathcal{L}} \ell_{\text{min},l} \Delta \mathbf{E}_l && (6.33)\\
\text{s.t.} \quad & \mathbf{E}_c[1] = E_c[0] + \sum_{l \in \text{In}(c)} (1 - \ell_{\text{min},l}) \Delta \mathbf{E}_l - \sum_{l \in \text{Out}(c)} (1 + \ell_{\text{max},l}) \Delta \mathbf{E}_l \\
& \mathbf{E}_c[1] = \mathbf{E}_{c+1}[1] \quad \forall c = 1, \ldots, N \\
& \Delta \mathbf{E}_l \geq 0
\end{aligned}
$$

In this LP formulation, we are interested in minimizing the sum of the losses that occur in all cells. This is equivalent to summing the losses that the transfers ΔE_l over all links entail using the link prices $\ell_{\text{min},l}$. The state of each cell, its energy, evolves linearly. This is owed to the re-formulated charge transfer dynamics (6.8). To ensure that cells move toward each other at the highest possible speed, we have transmitters incur the worst case losses ℓ_{max} while receivers only incur the best case ℓ_{min}. The goal is to have all cells at one level eventually; this is encoded by constraining each cell to equal its neighbor. As Problem (6.33) is a linear program, it can

be solved efficiently using a variety of off-line solvers. The following experiments have been conducted with CBC from the COIN-OR project [50] and its Python interface PuLP.

6.4.2 Constraint-driven charge routing

The performance bound from Section 6.4.1 is a useful tool to evaluate the room for improvement in routing strategies. With respect to this bound, heuristic strategies already perform well in scenarios without concurrency, as we will see in Section 6.4.3. This indicates that there may not be much to gain through further routing improvements. In this section, we therefore look for a strategy that remains above all simple with regards to computation, communication, and parallelization. Implementing heuristic strategies in parallel has typically lead to efficiency reductions previously, by contrast.

The solutions of the best case LP formulation (6.33) are often characterized entirely by the charge transfer direction that the main constraints indicate. This is intuitive because each transfer incurs a loss and one cannot expect to move charge in one direction and profitably reverse this transfer later on. Guided by this observation, the following paragraphs develop a routing law for architectures with transfers only between neighbors. As expected, this law turns out to be much simpler than a LP solver, even after adjustments for more complicated topologies. Nevertheless it yields excellent performance.

Basic implementation (assuming no losses) Assuming that transfers occur in a lossless fashion, the amount a cell must send or receive can be readily calculated. Using broadcasts or a daisy chain approach, the mean energy level $\bar{E}$ is calculated such that all participants are aware of their deviation $E_i - \bar{E}$. For each cell, this value represents how much its state has to be adjusted in total.

To start balancing, pick either boundary cell; $c = 1$ is chosen here. It is connected with the remainder of the pack only through a single link. This cell must therefore exchange the exact amount $E_1 - \bar{E}$ with its only neighbor. A regular cell c is connected with two neighbors, but it must already transfer a certain amount with its predecessor $c - 1$. The second cell's deviation for instance is given by $(E_2 + E_1 - \bar{E}) - \bar{E}$. These adjustments sum throughout the cell string and have the general form

$$\delta_{c,c+1} = \sum_{j=1}^{c} (E_j - \bar{E}). \tag{6.34}$$

Note that this calculation can be implemented efficiently in daisy chain fashion. We use $\delta_{c,c+1} = \Delta E_{c \to c+1}$ as a shorter notation for the energy that must be exchanged. Once all $\delta_{c,c+1}$ are calculated, each cell swaps exactly this amount with its neighbor. Ignoring the losses from these transfers, this procedure yields a perfectly balanced string. For cells $c < N$, this is clear by design. For N, the last cell, we find

$$E_N[1] = E_N[0] + \delta_{N-1} = E_N[0] + \sum_{j=1}^{N-1} (E_j[0] - \bar{E}) = \sum_{j=1}^{N} E_j[0] - (N-1)\bar{E} = \bar{E}. \tag{6.35}$$

There are multiple advantages in calculating the full transfer strategy upfront. We could slightly reduce dissipative losses, that typically dominate, by maximizing the interim voltages that occur along a chain of charge movements. For instance, if we have movements of the form $Q_1 \xrightarrow{q_1} Q_2 \xrightarrow{q_2} Q_3$, we can send q_1 before we send q_2. This increases the temporary voltage in Q_2 and hence reduces the losses during the second transfer. Yet, it also increases stress on cells and this trade-off should be carefully evaluated. Without further investigation, it appears prudent to divide $\delta_{c,c+1}$ into several blocks and schedule them such that they alternate with the transfers from other cells to limit the depth of discharge. The block size limit employed in the experiments of this chapter (Section 6.4.3) is clearly visible in the time series plot (Fig. 6.5).

Longer charge movements also reduce communication. Instead of arbitrating every few seconds, the constraint-driven approach can theoretically run for hours without recalculation or negotiation. In practice, minutes may be more realistic. This saves energy in practice and speeds up simulation which can also take larger steps without interrupting.

Anticipating losses The basic implementation of constraint-driven routing is sufficiently simple to run even on the cheapest embedded devices. However, it does not yet take into account the losses that the scheduled transfers entail. These losses depend on the individual cells; boundary cells in particular that either send or receive only as opposed to forwarding part of the incoming charge suffer significantly smaller losses. After concluding the scheduled transfers, we therefore do not end up with a perfectly balanced battery pack. In practice, we may be left with a deviation as high as $30\,\mathrm{bp}$ in SoC after starting balancing at an overall deviation of $300\,\mathrm{bp}$ as in Section 6.4.3. $1\,\mathrm{bp}$ is one basis point of deviation in SoC here, as introduced in Definition 5.3. Since the minimum SoC determines the energy level of the pack, this reduces the effectiveness of balancing significantly.

The end deviation remains elevated because the losses are entirely ignored. Using even a crude estimate, the performance can be improved to a satisfactory level (below $2\,\mathrm{bp}$ in Section 6.4.3). To that end, consider $\ell_{\mathrm{min},l}$ and $\ell_{\mathrm{max},l}$, the best and worst case loss ratios calculated for each link in Section 6.4.1. Using the mid-point of these predetermined boundaries, an estimate for the losses from the scheduled transfers can be calculated quickly. We define

$$\ell_{\mathrm{est},l,t} := \frac{1}{2}\left[\ell_{\mathrm{min},l,t} + \ell_{\mathrm{max},l,t}\right]$$

and analogously $\ell_{\mathrm{est},l,r}$ to gauge the losses of transmitting and receiving cell in a link.

Each transfer $\delta_{c,c+1}$ from the first set of transfers now yields loss estimates $\ell_{\mathrm{est},l,c}\delta_{c,c+1}$ and $\ell_{\mathrm{est},l,c+1}\delta_{c,c+1}$. Adjusting the state of all cells with these estimates and recalculating the transfers according to the basic version reduces the remaining end deviation. Since the losses change, as the charge transfers are adjusted, this process must be iterated multiple times for decent results. 5 iterations are typically sufficient, however.

This iterative approach keeps all advantages from calculating a priori as discussed for the basic version. Most noteworthy is that it does not increase the communication overhead. The additional computation time is moderate.

Constraint-driven approach in feedback loop If anticipating losses is not sufficient, constraint-driven routing can also operate with a feedback loop. By limiting the amount of energy

that can be sent at once, the process is separated into several iterations. After each iteration, affecting all participating cells, the charge transfers are reevaluated based on the new energy levels.

A feedback loop naturally increases communication overhead and energy threshold ΔE_{max} must thus be chosen sufficiently large. On the other hand, it should remain small to achieve the desired accuracy and prevent oscillations. As cell balancing is a slow process, however, finding a compromise should not be difficult.

Transfers between non-adjacent cells The routing technique from this section only deals with transfers between adjacent cells. This often leads to charge being transferred in daisy chain fashion. Direct transfers over a distance are almost always more efficient, however. Although resistances of neighbor transfers are generally lower due to fewer transistors and shorter distances, at least twice as many transfers are required. These lead to larger total losses in almost all cases. Accounting for the higher time requirement, and consequently using a lower current for non-neighbor transfers, makes the efficiency gain even more pronounced.

As transfers over distance are almost always preferable in ACB architectures that allow them, constraint-driven routing can be extended with the following heuristic. Instead of transferring all excess charge to the nearest neighbor, a cell shall meet the requirements of its neighbors one by one until its surplus is exhausted or the maximum transmission range is reached. In the latter case, the remaining surplus is transferred to the farthest cell within reach.

6.4.3 Routing case study

This section compares a heuristic strategy to the reference LP (Section 6.4.1) in order to gauge the room for further routing improvements. In the same experiment, constraint-driven routing (Section 6.4.2) demonstrates the benefits of parallel transfers and optimal current (Section 6.3).

The battery pack under consideration aims to mimic those from recent EVs like the Nissan Leaf or BMW i3 with a capacity of $21.6\,\text{kW h}$. It is arranged in a 96S24P fashion to reach both the desired capacity and operating voltage. This means, it consists of 96 series-connected cell units. Each unit contains 24 parallel-connected SAMSUNG INR18650-25R cells with a nominal voltage of $3.75\,\text{V}$ and a nominal capacity of $2.5\,\text{A h}$ [156]. These cells were characterized in Section 5.4 (Fig. 5.9 and Eq. (5.8)). Their typical internal resistance is $22.15\,\text{m}\Omega$. Recall that parallel-connected cells are considered as electrical unit from an ACB perspective since they balance on their own. Resistance and capacity of such a unit of 24 cells as well as inductor parameters are as follows.

$$\begin{bmatrix} R_C & Q_{\text{max}} \end{bmatrix} = \begin{bmatrix} \frac{1}{24} \times 22.15\,\text{m}\Omega & 24 \times 2.5\,\text{A h} \end{bmatrix} \qquad \begin{bmatrix} R_L & L \end{bmatrix} = \begin{bmatrix} 5\,\text{m}\Omega & 12\,\mu\text{H} \end{bmatrix}$$

These values correspond to inductors from the BOURNS 1140 series [24]. The transistor parameters were set according to the INFINEON OPTIMOS BSC010NE2LS data sheet [81].

$$\begin{bmatrix} R_M & \tau_u & \tau_d & C_{\text{oss}} \end{bmatrix} = \begin{bmatrix} 1.1\,\text{m}\Omega & 6.7\,\text{ns} + 6.0\,\text{ns} & 34\,\text{ns} + 4.4\,\text{ns} & 1700\,\text{pF} \end{bmatrix}$$

On a neighbor-only architecture (see Fig. 5.3), the following strategies are compared.

- BEST calculates a performance bound using LP formulation (6.33) for reference purposes.
- MAX is a heuristic strategy from [88]. It selects the strongest cell in terms of SoC and sends charge in the direction with lower mean SoC. After a so-called *macro step* of $30\,\mathrm{s}$, it calculates new transfers.
- ACD refers to anticipating constraint-driven routing as presented in Section 6.4.2.
- PAR is the parallel version of ACD. It constructs subsets for concurrent operation from the transfer list in a greedy fashion. This works well because the transfers are separated according to a block size limit $\Delta E_{\max}$ which ensures that they have similar amounts to process, besides the advantages discussed in Section 6.4.2.
- Strategies BEST*, ACD* and PAR* choose the optimal current dynamically via the ΔE-interface as explained in Section 5.5.3. This leads to higher efficiency but also increased balancing times.

Unless calculated dynamically, the peak current I is set to a value of $12\,\mathrm{A}$ which corresponds to about $\frac{1}{5}C$. This typical value ensures that the sequential strategies do not require unnecessarily long but do not dissipate too much energy either. It is larger than in other case studies from this thesis because the battery pack under consideration has a larger capacity. The transfers in this large-scale experiment have also been synchronized using a worst case cycle time. This simplifies actuation and simulation while increasing balancing time slightly but homogeneously.

The pack is considered to be equalized when the SoC deviation $\max_i\{z_i - \frac{1}{N}\sum_c z_c\}$ falls below $\epsilon = 0.1\,\mathrm{pp}$. This threshold is most critical for MAX where it serves as end condition. ACD and its derivatives achieve end deviations that are roughly an order of magnitude smaller by anticipating losses. For MAX to achieve the same without long and wasteful oscillations during the final stages of balancing the macro step would have to be inconveniently small.

The strategies are compared in terms of speed or balancing time as well as in terms of losses. To make the loss terms more tangible, we calculate them with respect to the maximum possible energy that can be gained through balancing. Consider a string of cells with energies E_c. The amount of energy we can draw from the pack without balancing is given by $N \cdot \min_c E_c$. If we were to balance in a lossless fashion on the other hand, we could draw $\sum_c E_c$. After obtaining the losses, via simulation or from optimization, the loss ratio is calculated as

$$\text{loss ratio} = \frac{\text{losses}}{\sum_c E_c - N \cdot \min_c E_c}. \tag{6.36}$$

Passive balancing corresponds to a loss ratio of $100\,\%$ since all cell energies are reduced to the weakest cell by dissipating excess charge.

The battery cells are initialized with an SoC of $z_c = z_{\mu,s} + z_{c,s}$. In each scenario s, the lowest SoC $z_\mu \sim \text{Uniform}(0.3, 0.6)$ is generated first. The individual $z_{c,s} \sim \text{Uniform}(0.00, 0.03)$ is added subsequently. In other words, the analyzed SoC distributions are spread out over at most $3\,\mathrm{pp}$ and occur roughly within the second third of the SoC spectrum.

Fig. 6.4 shows an overview from 20 such scenarios. Comparing ACD and BEST (or ACD* and BEST*), we find that the routing selected by ACD is indeed highly efficient. It may in fact be

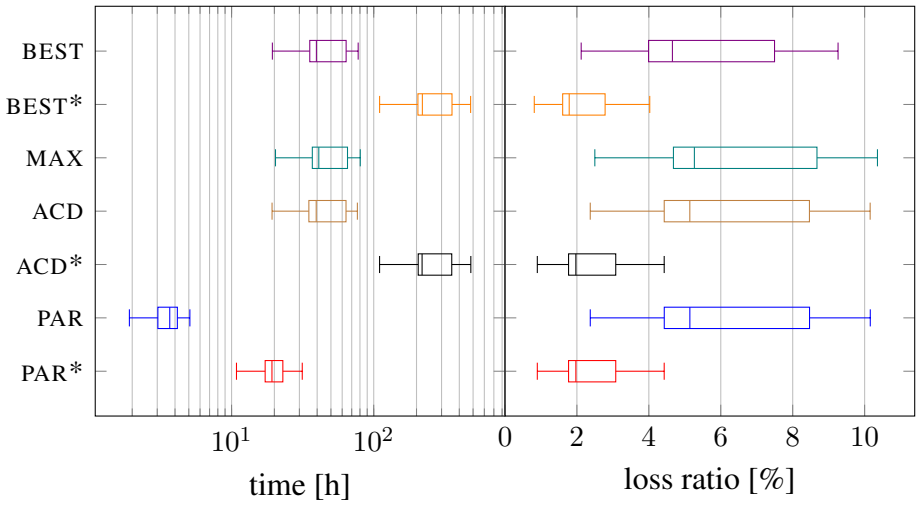


Figure 6.4: *Both* MAX *and the proposed* ACD *strategy operate close to the unachievable reference bound* BEST *when facing random initial SoC distributions. This indicates small or no potential for further gains from smarter charge routing. Optimal current choice (starred versions) reduces losses significantly but requires more time. Parallelization is thus essential.*

optimal since BEST calculates with optimistic assumptions that are not achievable in practice. That said, the heuristic routing selected by MAX also leads to decent results. This strategy dissipates only $3\,\%$ more than ACD. Calculating transfers a priori as in ACD saves additional energy in reduced communication that is not reflected here, however.

With heuristic (MAX) and constraint-driven routing (ACD) performing close to the optimum (BEST), there is almost nothing to be gained from further investigating routing strategies. Large efficiency gains come only from selecting a more suitable current. Operating at $I \approx 2.3\,\mathrm{A}$, ACD* loses less than 40% of the fixed current alternatives. These variations trade off speed however and the balancing times they require (weeks for the scenarios here) may be infeasible in practice.

A remedy for the long balancing durations is parallelization which is another strength of the constraint-driven approach. In MAX and similar heuristic algorithms parallelization is non-obvious. Attempts that also send from the second strongest, third strongest cell, etc., have always traded off efficiency for concurrency, for instance. With PAR, we do not trade off efficiency (see Fig. 6.4). Even though it constructs the parallel movements in a greedy fashion, it dissipates only fractions of a percent more energy than ACD. It is, however, 12 times faster on average. Accelerated in this way, PAR* is both faster and more efficient than ACD and MAX. Fig. 6.5 shows how the individual SoC values evolve over time in one of the examined scenarios. There, MAX drives down the maximum value towards the average as expected. ACD and PAR* perform similar charge movements. The first transfers at a higher rate, but sequentially. The short bumps in SoC occur when a cell forwards charge such as in the middle of a transfer chain $2 \to 3 \to 4$.

In some cases, the speed achieved via parallelization may still be insufficient. In these cases, the transfer rate, i.e., the peak current must be increased. Fig. 6.6 illustrates the trade-off in this situation. Raising the current away from the optimal value increases the loss ratio. This leads

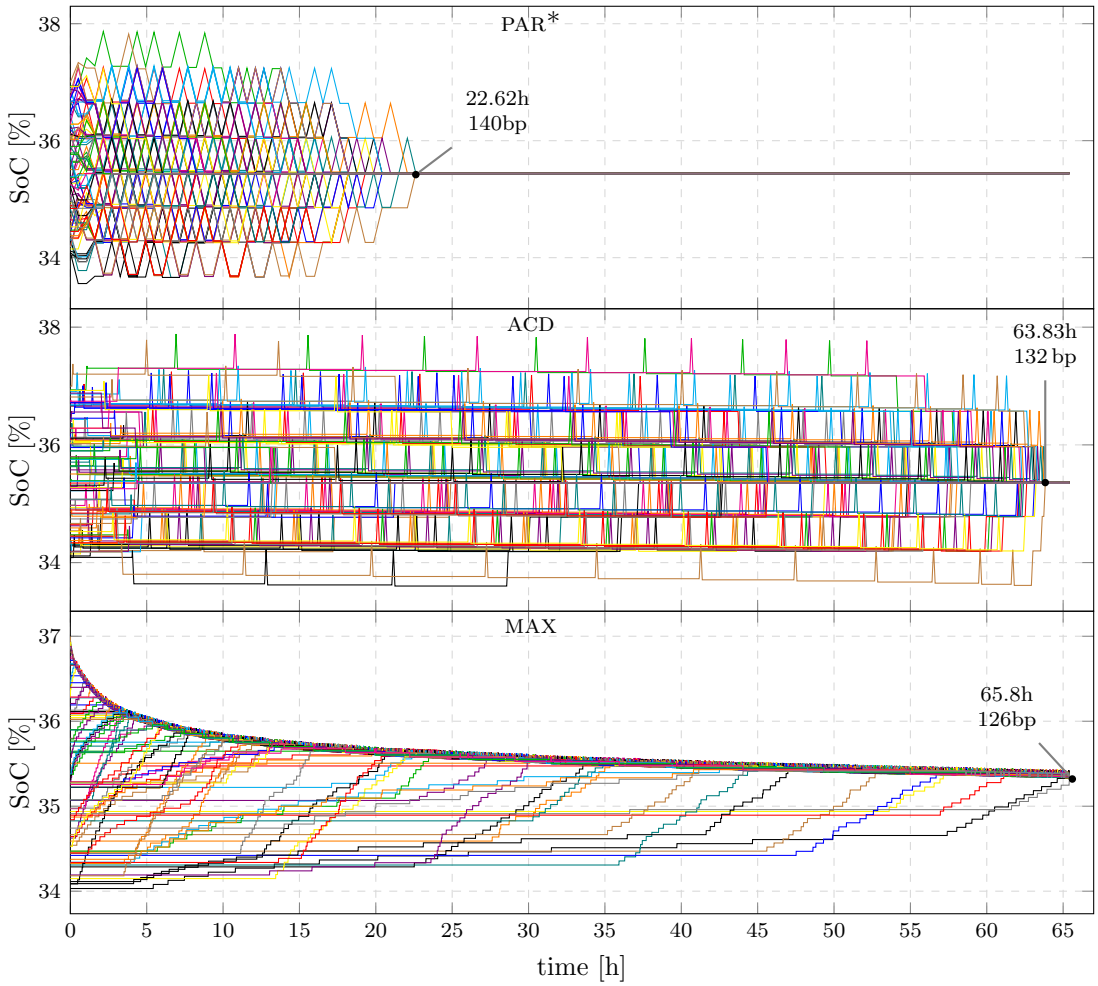


Figure 6.5: *The constraint-driven approach parallelizes so naturally that* PAR*, *running at the lower, optimal current is both faster and more efficient than its sequential version* ACD *and state-of-the-art strategy* MAX *operating at higher current. Here, it increases the pack SoC* 11 % *more (*140 bp *vs* 126 bp*) than* MAX *in less than half the time.*

to slow increments at first and balancing times under 4 h are achievable while dissipating less than 10 % of the energy gained via balancing. In some scenarios even balancing times around 1 h are possible. Beyond that, the efficiency deteriorates rapidly however. At loss ratios over 20 %, the loss estimation process behind the ACD strategy becomes more difficult. Since these values are also increasingly meaningless, we do not simulate in this parameter range.

6.5 Related work

In this section, the literature for ACB optimization is discussed. This includes modeling techniques, topology comparisons that are typically based on these models, a priori parameterization and runtime strategies.

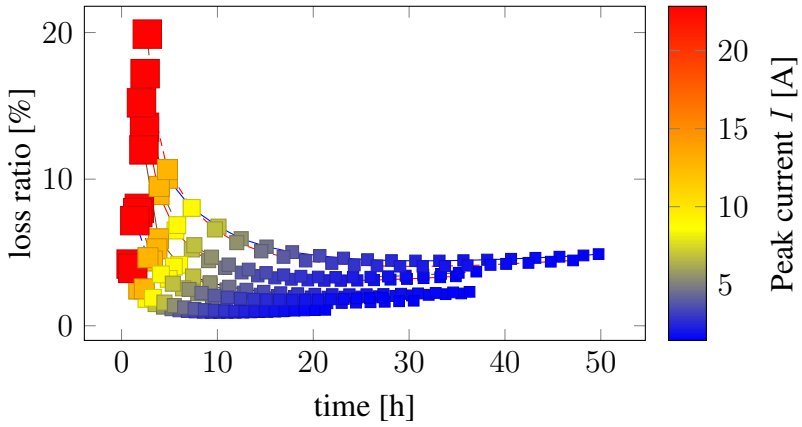


Figure 6.6: *If parallelization alone is not sufficient, we can trade off efficiency for speed by increasing the peak current I that we utilize in strategy* PAR.

Modeling for optimization In the control domain, it is typically assumed that the effective balancing current can be freely chosen. This transforms the complex problem into a linear program, but it ignores actuation aspects and the effects from changing cell voltages. As the voltage of the receiving cell rises, for instance, phase timing must be shortened to maintain a fixed current which reduces the transferred charge. The inaccuracies of such models are shown in Fig. 6.1 on page 127. As they lead to simple, efficient computation, they are nevertheless used in the literature to compare ACB topologies [34, 36, 149], among other things. These comparisons are further discussed in an upcoming paragraph. [51], by contrast, investigates the problem class in general under these linearity assumptions and finds that it can be solved by MPC with a horizon of one step.

[35] presents an approach that is similar to the I-interface (Section 5.5.2). The authors introduce a linear voltage slope and discuss frequency adjustments to maintain a specified current because "[with constant frequency], a change from the voltage level $4.2\,\mathrm{V}$ to $2.7\,\mathrm{V}$ leads to a decrease of current by $35\,\%$" [35]. They include this behavior into a mathematical program which trades off cell imbalances with actuation losses and total balancing time, as in many MPC formulations. To solve that program with its nonlinearities, they resort to Dynamic Programming (DP) and use direct collocation. Since DP resorts to a discrete grid, this approach may have limited scalability; the case study only considers 8 cells.

A detailed convex modeling approach for batteries in hybrid EV is presented in [136]. This paper uses and justifies a piecewise-linear voltage model to consider the load dynamics of a battery in terms of energy and power, similar to the thesis at hand (Section 6.1). [136] deals with battery sizing as well as load distribution between battery and combustion engine, however, and not with ACB. For this reason, no switching dynamics are considered. As the currents are also higher, the reported relative errors are above $10\,\%$ in many cases.

Topology evaluation Section 5.9 qualitatively describes the many topologies for ACB, like Cell Bypass, Cell to Pack, Pack to Cell, and Cell to Cell. While this thesis focuses on Cell to Cell balancing, other works deal with the quantitative comparison of the various topologies.

Caspar, for instance, compares various ACB approaches assuming constant transfer effi-

ciency. In [34], he analyzes a battery consisting of 96 cells, which is subdivided into isolated modules with 8 cells . He reports that single inductor and flyback converter architectures lead to the longest driving distance in the two scenarios whereas a Ćuk converter balancing architecture achieves the fastest time. In [36], he uses optimal control to make the comparison more systematic. Simulating up to 14 cells, it is shown that direct Cell to Cell leads to the lowest transfer losses, but relatively high balancing times. Having a fully connected graph significantly increases speed with minor additional losses, but requires more components.

Baronti models voltage dependency, but also assumes constant transfer efficiency to perform statistical simulations of an inductor-based architecture with 10 cells and varying topology. In this way, he finds that Cell to Cell with remote transfers "outperforms the other methods in terms of both balancing time and energy loss" [16].

Preindl goes beyond simulation and compares ACB topologies based on their worst case performance in [149]. To that end, a LP formulation is evaluated for all vertices of a polytopic set of initial imbalances. A convexity argument justifies that this yields the worst case. Assuming identical link efficiency and ignoring concurrency as well as the trade-off between losses and speed, his analysis finds that Cell to Pack approaches dominate in terms of balancing time and efficiency.

Architecture optimization After topology and transfer mechanism are decided, there are still several design issues that determine the efficiency of an ACB application: offline optimization of components and switching network and strategies for control at runtime. While the latter is addressed by several contributions in the literature as discussed in the next paragraph, the former leads to more complicated optimization and is still emerging. The author of this thesis and immediate colleagues have investigated the systematic selection of off-the-shelf [138] and the design of an optimal inductor [89] (Section 6.2). A design methodology for the transistor network and the associated switching scheme has been presented in [122]. Extending the SAT-based verification from [123], this work significantly reduces the transistor and signal count for non-adjacent Cell to Cell transfers.

Charge routing strategies Actuation at runtime is another important factor for ACB performance. Papers that compare topologies implicitly deal with optimal routing as well [36,51,149]. As they often select constant currents a priori and assume fixed link efficiencies, they are typically not accurate enough to justify their computational cost, particularly in embedded environments. Several heuristic strategies have hence been proposed in the literature. In [111], a balancing strategy using fuzzy control is presented and evaluated using SPICE simulation, however without considering pack-level criteria. Several request-driven strategies for inductor-based balancing of battery packs are discussed in [166]. An initial, fully distributed algorithm for charge equalization of super-capacitors is presented in [117]. Many of the balancing architectures discussed in Section 5.9 also propose their own actuation strategies.

7 Conclusions and Future Work

Conclusions

This thesis studies semantic gaps in digital control between high-level models for the physical process and those for the underlying computation platform. In the first part, it analyzes how a platform and a fast process are linked by delay which is typically either ignored or abstracted too coarsely. The second part deals with switching actuation in the case of Active Cell Balancing (ACB). There, the timing-based input is typically abstracted to current in ways that ignore important details. Since monolithic models lead to unacceptable computation times in both cases, suitable interfaces are necessary to describe the behavior of the physical side in a form that can be taken into account in the analysis of the digital side and vice versa. Such interfaces are proposed in the scope of this thesis. In the first part, they lead to improvements in terms of guaranteed control quality under communication constraints by considering delay patterns with more detail. In the second part, they speed up accurate simulation by orders of magnitude and form the basis for optimization techniques.

The main contributions and results of this work can be summarized as follows.

- **Fixed-Priority Non-preemptive Scheduling (FPNS) in the ECA framework**

  The ECA framework is based on discrete time steps that are typically much longer than the arbitration times in FPNS to keep computation feasible. A more realistic time step is in the range of message transmission time. Many other automaton frameworks share this situation. While it is not an issue for time-triggered communication where the time steps can be aligned with the message transmissions, event-triggered approaches like FPNS require the analysis of messages that arrive in the middle of a time step.

  Chapter 3 explains how straightforward FPNS modeling leads to overly optimistic delay bounds that are often broken in practice. This may have catastrophic outcomes, like a failure of the anti-lock braking system, if it is not discovered during testing. The chapter

thus proceeds to take the non-deterministic behaviors produced by the message arrivals into account during model checking to obtain conservative but safe timing guarantees.

- **Delay pattern interface for CPS co-design and fault-tolerant control strategy**

  During controller design, the digital implementation platform is typically either considered ideal, i.e., with zero delay or reduced to its worst-case delay. For the platform design, on the other hand, the controller requirements are given as deadlines that specify the acceptable delay. This leads to suboptimal designs since the worst-case delay is rare in most platforms and significantly exceeds the average.

  Chapter 4 demonstrates how delay patterns can be used as more expressive interface. They can be checked on an ECA model of the implementation platform and translate to various control criteria. In addition, knowledge of these patterns leads to a fault-tolerant control strategy that takes deadline misses into account at runtime and achieves exponential stability by construction. Overall, this interface tightens the integration of platform and controller to yield improved designs.

- **Computationally efficient, but accurate models for Active Cell Balancing (ACB)**

  Efficient charge transfer methods for ACB are actuated by varying the durations of alternating switch configurations. This is challenging for design automation since nonlinear behavior and non-differentiable phase transitions, that require high resolution, complicate optimization and even lead to slow simulation. For this reason, the switching details are often abstracted away and replaced by average current. Although simpler, this also forgoes accuracy. Detailed computation, conversely, can also remain efficient.

  Chapter 5 demonstrates how the transfer dynamics of each phase can be solved in closed form to yield a recurrence relation that does not require a numeric solver. Based on this representation, several lossless actuation interfaces are developed that formalize the interaction over the long term and enable further reformulations. The latter lead to simulation techniques which introduce virtually no error and achieve speedups of up to 5 orders of magnitude, that are vital for interactive design and bulk evaluation of multiple scenarios.

- **Optimal parameterization and actuation of ACB architectures**

  While fast simulations can lead to good designs via interactive iterations, optimization via mathematical programming leads to additional insights and a higher degree of automation. For this reason, Chapter 6 continues the bottom-up modeling from Chapter 5 but forgoes some accuracy, mostly in inefficient and thus less important operating regions, to obtain a formulation that is compatible with efficient optimization techniques. This formulation is used to design optimal inductors for ACB that outperform off-the-shelf candidates because the latter make provisions for unnecessarily high currents. Furthermore, it leads to a best-case reference bound for charge routing to guide future research endeavors. Non-concurrent heuristic strategies already perform well relative to this bound, motivating the development of a strategy that focuses on simplicity and parallelization.

Notwithstanding the value of these contributions in their respective situations, careful analysis is recommended to check whether the circumstances necessitate such sophisticated techniques.

Providing a more powerful platform is often an alternative to the techniques presented in the first part of this thesis. The higher costs this leads to may be acceptable since processors and network bandwidth have grown over the years while control transmissions and computations have often remained short.

ACB, from the second part, needs to prove that it should be substituted for passive balancing, the current state of the art. This depends on factors like pack size, cell variation, component costs, and transfer efficiency. Energy savings which can be obtained from imbalance measurements and simulation techniques, like those proposed in the thesis at hand, must be weighed with the associated costs.

Future work

Even though the proposed techniques significantly improve existing interfaces for CPS and ACB design, many questions remain. Some promising directions for future research are pointed out in the following paragraph. Potential issues should be investigated to increase robustness, but there is also untapped performance.

- **Robustness analysis for message losses**

  A big strength of feedback control systems is their inherent robustness. Good performance can be achieved even when model parameters are not perfectly known, signals are delayed, or when there are external disturbances. However, these factors may not be independent of each other. In other words, we may trade in robustness to model uncertainties for robustness to communication faults. For this reason, it appears prudent to investigate how other factors are affected when the system must compensate for message losses.

- **State estimation to alleviate message losses**

  When an actuator does not receive new input information, it has several options. The most common options are to reapply the current input or to apply no input at all. If it is possible to perform small computations in case of a message loss, however, state estimators become an option. Such constellations are explored in event-triggered control where some messages are deliberately not sent and may also mitigate communication faults in platforms where computation is less scarce.

- **Inductor design with more flexible input current**

  The inductor design presented in Chapter 6 can be extended in several ways. For instance, it may be possible to model more behavior as variable and not with respect to the worst-case voltages. More importantly, the transfer current could be created in a more flexible way. Instead of operating between zero and a peak value, a non-zero trough current could be used. This reduces the peak current required to achieve a specified speed and may improve efficiency.

- **ACB against cell aging**

  ACB is mainly considered from an efficiency perspective. However, ACB also alleviates stress on weaker cells and may hence increase their lifetime. Further investigation is needed to determine whether the overall pack benefits from this.

- **Battery characterization experiments at ACB frequencies**

  Current cell models are mainly designed for SoC estimation and simulation of device operation. The conditions there are different from those in ACB. Even though the models capture pulses in the millisecond range that an electric motor may produce, they are not tested on triangle waves in the kilohertz range with positive offset. If such experiments ever produce more detailed models, the analysis from this thesis can be amended to include them as long as a closed-form solution for the intra-phase dynamics can be found, as in Section 5.7. This is possible, e.g., for second-order models.

In a broader scope, it is desirable to integrate switching questions into formal methods for CPS analysis. Tools like SpaceEx [66] and Pessoa [130] have recently made significant inroads in this field. As pointed out throughout this thesis, they cannot include the full details of communication hardware or switching actuation directly, however. Nevertheless, a formal verification appears feasible with appropriate interfaces and some adjustments. Such an integration could lead to further automation and a more systematic approach to CPS design.

Bibliography

[1] A123 SYSTEMS, *High Power Lithium Ion APR18650M1*, 2008. `www.a123batteries.com/v/vspfiles/images/pdf/APR18650M1A.pdf`.

[2] A. ALBERT, *Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems*, in Proc. of Embedded World, 2004.

[3] M. ALTHOFF, *An Introduction to CORA 2015*, in Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), 2015.

[4] R. ALUR AND D. L. DILL, *A Theory of Timed Automata*, Theoretical Computer Science, 126 (1994), pp. 183–235.

[5] R. ALUR AND G. WEISS, *Regular Specifications of Resource Requirements for Embedded Control Software*, in Proc. of Real-Time and Embedded Technology and Applications Symposium (RTAS), 2008.

[6] D. ANDREA, *Battery Management Systems for Large Lithium Ion Battery Packs*, Artech House, 2010.

[7] A. ANTA, R. MAJUMDAR, I. SAHA, AND P. TABUADA, *Automatic Verification of Control System Implementations*, in Proc. of International Conference on Embedded Software (EMSOFT), 2010.

[8] A. ANTA AND P. TABUADA, *To sample or not to sample: Self-triggered control for nonlinear systems*, IEEE Transactions on Automatic Control, 55 (2010), pp. 2030–2042.

[9] E. S. ARMSTRONG AND A. K. CAGLAYAN, *An Algorithm for the Weighting Matrices in the Sampled-data Optimal Linear Regulator Problem*, NASA Tech. Note, TN D-8372 (1976).

[10] K. J. ÅSTRÖM AND B. WITTENMARK, *Computer-Controlled Systems: Theory and Design*, Courier Dover Publications, 2011.

[11] M. ATHANS, *The Role and Use of the Stochastic Linear-Quadratic-Gaussian Problem in Control System Design*, IEEE Transactions on Automatic Control, 16 (1971), pp. 529–552.

[12] C. BAIER AND J.-P. KATOEN, *Principles of Model Checking*, MIT Press, 2008.

[13] F. BARONTI, C. BERNARDESCHI, L. CASSANO, A. DOMENICI, R. RONCELLA, AND R. SALETTI, *Design and Safety Verification of a Distributed Charge Equalizer for Modular Li-Ion Batteries*, IEEE Transactions on Industrial Informatics, 10 (2014), pp. 1003–1011.

[14] F. BARONTI, G. FANTECHI, R. RONCELLA, AND R. SALETTI, *Intelligent Cell Gauge for a Hierarchical Battery Management System*, in Proc. of IEEE Transportation Electrification Conference (ITEC), 2012.

[15] ——, *High-Efficiency Digitally Controlled Charge Equalizer for Series-Connected Cells Based on Switching Converter and Super-Capacitor*, IEEE Transactions on Industrial Informatics, 9 (2013), pp. 1139–1147.

[16] F. BARONTI, R. RONCELLA, AND R. SALETTI, *Performance Comparison of Active Balancing Techniques for Lithium-Ion Batteries*, Journal of Power Sources, 267 (2014), pp. 603–609.

[17] R. H. BARTELS AND G. W. STEWART, *Solution of the Matrix Equation AX + XB = C [F4]*, Communications of the ACM, 15 (1972), pp. 820–826.

[18] A. BAUGHMAN AND M. FERDOWSI, *Double-Tiered Switched-Capacitor Battery Charge Equalization Technique*, IEEE Transactions on Industrial Electronics, 55 (2008), pp. 2277–2285.

[19] T. BAUMHÖFER, M. BRÜHL, S. ROTHGANG, AND D. U. SAUER, *Production Caused Variation in Capacity Aging Trend and Correlation to Initial Cell Performance*, Journal of Power Sources, 247 (2014), pp. 332–338.

[20] G. BEHRMANN, A. DAVID, AND K. G. LARSEN, *A Tutorial on Uppaal*, in Formal Methods for the Design of Real-Time Systems, Springer, 2004, pp. 200–236.

[21] D. BERTSEKAS, *Dynamic Programming and Optimal Control*, vol. 1, Athena Scientific, 1995.

[22] P. BOGACKI AND L. F. SHAMPINE, *A 3(2) pair of Runge - Kutta Formulas*, Applied Mathematics Letters, 2 (1989), pp. 321–325.

[23] A. BOUILLARD, L. PHAN, AND S. CHAKRABORTY, *Lightweight Modeling of Complex State Dependencies in Stream Processing Systems*, in Proc. of Real-Time and Embedded Technology and Applications Symposium (RTAS), 2009.

[24] BOURNS, INC., *High Current Chokes*, 2014. `www.bourns.com/data/global/pdfs/1140_series.pdf`.

[25] S. BOYD, S.-J. KIM, L. VANDENBERGHE, AND A. HASSIBI, *A Tutorial on Geometric Programming*, Optimization and Engineering, 8 (2007), pp. 67–127.

[26] S. P. BOYD, L. EL GHAOUI, E. FERON, AND V. BALAKRISHNAN, *Linear Matrix Inequalities in System and Control Theory*, SIAM, 1994.

[27] M. BRANDL, H. GALL, M. WENGER, V. LORENTZ, M. GIEGERICH, F. BARONTI, G. FANTECHI, L. FANUCCI, R. RONCELLA, R. SALETTI, ET AL., *Batteries and Battery Management Systems for Electric Vehicles*, in Proc. of Design, Automation and Test in Europe (DATE), 2012.

[28] M. BRANICKY, S. PHILLIPS, AND W. ZHANG, *Scheduling and Feedback Co-Design for Networked Control Systems*, in Proc. of Conference on Decision and Control (CDC), 2002.

[29] M. S. BRANICKY, *Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems*, IEEE Transactions on Automatic Control, 43 (1998), pp. 475–482.

[30] M. S. BRANICKY, S. M. PHILLIPS, AND W. ZHANG, *Stability of Networked Control Systems: Explicit Analysis of Delay*, in Proc. of American Control Conference (ACC), 2000.

[31] S. BULLER, M. THELE, R. W. A. A. DE DONCKER, AND E. KARDEN, *Impedance-Based Simulation Models of Supercapacitors and Li-ion Batteries for Power Electronic Applications*, IEEE Transactions on Industrial Applications, 41 (2005), pp. 742–747.

[32] J. CAO, N. SCHOFIELD, AND A. EMADI, *Battery Balancing Methods: A Comprehensive Review*, in Proc. of Vehicle Power and Propulsion Conference (VPPC), 2008.

[33] J. R. CASH AND A. H. KARP, *A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides*, ACM Transactions on Mathematical Software, 16 (1990), pp. 201–222.

[34] M. CASPAR, T. EILER, AND S. HOHMANN, *Comparison of Active Battery Balancing Systems*, in Proc. of Vehicle Power and Propulsion Conference (VPPC), 2014.

[35] M. CASPAR AND S. HOHMANN, *Optimal Cell Balancing with Model-based Cascade Control by Duty Cycle Adaption*, in Proc. of IFAC World Congress, 2014.

[36] M. CASPAR, V. POLINSKI, AND S. HOHMANN, *Structural Comparison of Battery Balancing Architectures with Optimal Control*, in Proc. of Vehicle Power and Propulsion Conference (VPPC), 2015.

[37] R. CASTANE, P. MARTI, M. VELASCO, A. CERVIN, AND D. HENRIKSSON, *Resource Management for Control Tasks Based on the Transient Dynamics of Closed-Loop Systems*, in Proc. of Euromicro Conference on Real-Time Systems (ECRTS), 2006.

[38] A. CERVIN, *Using Jitterbug to Derive Control Loop Timing Requirements*, in Proc. of Co-design for Embedded Real-time Systems (CERTS), 2003.

[39] S. CHAKRABORTY, S. KÜNZLI, AND L. THIELE, *A General Framework for Analysing System Properties in Platform-Based Embedded System Designs*, in Proc. of Design, Automation and Test in Europe (DATE), 2003.

[40] S. CHAKRABORTY, S. KÜNZLI, L. THIELE, A. HERKERSDORF, AND P. SAGMEISTER, *Performance Evaluation of Network Processor Architectures: Combining Simulation with Analytical Estimation*, Computer Networks, 41 (2003), pp. 641–665.

[41] S. CHAKRABORTY, L. PHAN, AND P. THIAGARAJAN, *Event Count Automata: A State-Based Model for Stream Processing Systems*, in Proc. of Real-Time Systems Symposium (RTSS), 2005.

[42] R. N. CHARETTE, *This Car Runs on Code*, 2009. `http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code`.

[43] N. CHATURVEDI, R. KLEIN, J. CHRISTENSEN, J. AHMED, AND A. KOJIC, *Algorithms for Advanced Battery-Management Systems*, IEEE Control Systems Magazine, 30 (2010), pp. 49–68.

[44] J. CHATZAKIS, K. KALAITZAKIS, N. C. VOULGARIS, AND S. N. MANIAS, *Designing a New Generalized Battery Management System*, IEEE Transactions on Industrial Electronics, 50 (2003), pp. 990–999.

[45] M. CHEN AND G. RINCON-MORA, *Accurate Electrical Battery Model Capable of Predicting Runtime and I-V Performance*, IEEE Transactions on Energy Conversion, 21 (2006), pp. 504–511.

[46] C.-F. CHIASSERINI AND R. R. RAO, *Energy Efficient Battery Management*, IEEE Journal on Selected Areas in Communications, 19 (2001), pp. 1235–1245.

[47] D. B. CHOKSHI AND P. BHADURI, *Modeling Fixed Priority Non-Preemptive Scheduling with Real-Time Calculus*, in Proc. of IEEE Intl. Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2008.

[48] F. CIUCU, *The Stochastic Network Calculus: A Modern Approach to Queueing Using Inequalities*, in Proc. of Asian Internet Engineering Conference (AINTEC), 2009.

[49] F. CIUCU AND J. SCHMITT, *Perspectives on Network Calculus: No Free Lunch, but Still Good Value*, ACM SIGCOMM Computer Communcation Review, 42 (2012), pp. 311–322.

[50] COIN-OR FOUNDATION, *COmputational INfrastructure for Operations Research*. `http://www.coin-or.org/`.

[51] C. DANIELSON, F. BORRELLI, D. OLIVER, D. ANDERSON, AND T. PHILLIPS, *Constrained Flow Control in Storage Networks: Capacity Maximization and Balancing*, Automatica, 49 (2013), pp. 2612–2621.

[52] M. DAOWD, N. OMAR, P. VAN DEN BOSSCHE, AND J. VAN MIERLO, *Passive and Active Battery Balancing Comparison Based on MATLAB Simulation*, in Proc. of Vehicle Power and Propulsion Conference (VPPC), 2011.

[53] R. I. DAVIS, A. BURNS, R. J. BRIL, AND J. J. LUKKIEN, *Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised*, Real-Time Systems, 35 (2007), pp. 239–272.

[54] L. DE MOURA, S. OWRE, H. RUESS, J. RUSHBY, N. SHANKAR, M. SOREA, AND A. TIWARI, *SAL 2*, in Proc. of Computer Aided Verification (CAV), 2004.

[55] D. W. DEES, V. S. BATTAGLIA, AND A. BÉLANGER, *Electrochemical Modeling of Lithium Polymer Batteries*, Journal of Power Sources, 110 (2002), pp. 310–320.

[56] J. R. DORMAND AND P. J. PRINCE, *A Family of Embedded Runge-Kutta Formulae*, Journal of Computational and Applied Mathematics, 6 (1980), pp. 19–26.

[57] M. DUBARRY, B. Y. LIAW, M.-S. CHEN, S.-S. CHYAN, K.-C. HAN, W.-T. SIE, AND S.-H. WU, *Identifying Battery Aging Mechanisms in Large Format Li-Ion Cells*, Journal of Power Sources, 196 (2011), pp. 3420–3425.

[58] M. DUBARRY, N. VUILLAUME, AND B. Y. LIAW, *Origins and Accommodation of Cell Variations in Li-ion Battery Pack Modeling*, International Journal of Energy Research, 34 (2010), pp. 216–231.

[59] M. DWYER, G. AVRUNIN, AND J. CORBETT, *Patterns in Property Specifications for Finite-State Verification*, in Proc. of International Conference on Software Engineering (ICSE), 1999.

[60] M. EINHORN, W. ROESSLER, AND J. FLEIG, *Improved Performance of Serially Connected Li-Ion Batteries With Active Cell Balancing in Electric Vehicles*, IEEE Transactions on Vehicular Technology, 60 (2011), pp. 2448–2457.

[61] J. EKER, P. HAGANDER, AND K.-E. ÅRZÉN, *A Feedback Scheduler for Real-Time Controller Tasks*, Control Engineering Practice, 8 (2000), pp. 1369–1378.

[62] J. EKER, J. W. JANNECK, E. LEE, J. LIU, X. LIU, J. LUDVIG, S. NEUENDORFFER, S. SACHS, Y. XIONG, AND OTHERS, *Taming Heterogeneity - the Ptolemy Approach*, Proceedings of the IEEE, 91 (2003), pp. 127–144.

[63] R. W. ERICKSON AND D. MAKSIMOVIC, *Fundamentals of Power Electronics*, Springer, 2001.

[64] E. FERON AND F. ALEGRE, *Control Software Analysis, Part I Open-loop Properties*, arXiv:0809.4812, (2008). `http://arxiv.org/abs/0809.4812`.

[65] E. FERSMAN, L. MOKRUSHIN, P. PETTERSSON, AND W. YI, *Schedulability Analysis of Fixed-Priority Systems using Timed Automata*, Theoretical Computer Science, 354 (2006), pp. 301–317.

[66] G. FREHSE, C. L. GUERNIC, R. DONZÉ, S. COTTON, R. RAY, O. LEBELTEL, R. RIPADO, A. GIRARD, AND T. DANG, *SpaceEx: Scalable Verification of Hybrid Systems*, in Proc. of Computer Aided Verification (CAV), 2011.

[67] B. FRIEDLAND, *Control System Design: An Introduction to State-Space Methods*, Courier Corporation, 2012.

[68] H. FUJIOKA AND T. NAKAI, *Stabilising Systems with Aperiodic Sample-and-Hold Devices: State Feedback Case*, IET Control Theory & Applications, 4 (2010), pp. 265–272.

[69] J. GALLARDO-LOZANO, E. ROMERO-CADAVAL, M. I. MILANES-MONTERO, AND M. A. GUERRERO-MARTINEZ, *Battery equalization active methods*, Journal of Power Sources, 246 (2014), pp. 934–949.

[70] J. GARCHE AND A. JOSSEN, *Battery Management Systems (BMS) for Increasing Battery Life Time*, in Proc. of Telecommunications Energy Special Conference (TELESCON), 2000.

[71] A. GERSTLAUER, C. HAUBELT, A. PIMENTEL, T. STEFANOV, D. GAJSKI, AND J. TEICH, *Electronic System-Level Synthesis Methodologies*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28 (2009), pp. 1517–1530.

[72] R. GOGOANA, M. B. PINSON, M. Z. BAZANT, AND S. E. SARMA, *Internal Resistance Matching for Parallel-Connected Lithium-ion Cells and Impacts on Battery Pack Cycle Life*, Journal of Power Sources, 252 (2014), pp. 8–13.

[73] M. GRANT AND S. BOYD, *CVX: Matlab Software for Disciplined Convex Programming*, 2014. `http://cvxr.com/cvx`.

[74] H. HE, R. XIONG, X. ZHANG, F. SUN, AND J. FAN, *State-of-Charge Estimation of the Lithium-Ion Battery Using an Adaptive Extended Kalman Filter Based on an Improved Thevenin Model*, IEEE Transactions on Vehicular Technology, 60 (2011), pp. 1461–1469.

[75] D. HENRIKSSON AND A. CERVIN, *Optimal On-line Sampling Period Assignment for Real-Time Control Tasks Based on Plant State Information*, in Proc. of Conference on Decision and Control (CDC), 2005.

[76] E. HENRIKSSON, H. SANDBERG, AND K. H. JOHANSSON, *Predictive Compensation for Communication Outages in Networked Control Systems*, in Proc. of Conference on Decision and Control (CDC), 2008.

[77] J. HESPANHA, P. NAGHSHTABRIZI, AND Y. XU, *A Survey of Recent Results in Networked Control Systems*, Proceedings of the IEEE, 95 (2007), pp. 138 –162.

[78] X. HU, F. SUN, AND Y. ZOU, *Estimation of State of Charge of a Lithium-Ion Battery Pack for Electric Vehicles Using an Adaptive Luenberger Observer*, Energies, 3 (2010), pp. 1586–1603.

[79] K. HUANG, L. THIELE, T. STEFANOV, AND E. DEPRETTERE, *Performance Analysis of Multimedia Applications using Correlated Streams*, in Proc. of Design, Automation and Test in Europe (DATE), 2007.

[80] A. IMTIAZ AND F. KHAN, *"Time Shared Flyback Converter" Based Regenerative Cell Balancing Technique for Series Connected Li-Ion Battery Strings*, IEEE Transactions on Power Electronics, 28 (2013), pp. 5960–5975.

[81] INFINEON TECHNOLOGIES AG, *BSC010NE2LS - OptiMOS Power-MOSFET*, 2013. `https://www.infineon.com/dgdl/Infineon-BSC010NE2LS-DS-v02_02-en.pdf?fileId=db3a304326dfb1300126fb3d176a3f1b`.

[82] M. ISAACSON, R. HOLLANDSWORTH, P. GIAMPAOLI, F. LINKOWSKY, A. SALIM, AND V. TEOFILO, *Advanced Lithium Ion Battery Charger*, in Proc. of Battery Conference on Applications and Advances, 2000.

[83] K. JENSEN, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. 1, Springer, 2013.

[84] K. JENSEN, L. M. KRISTENSEN, AND L. WELLS, *Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems*, International Journal on Software Tools for Technology Transfer, 9 (2007), pp. 213–254.

[85] Y. JIANG, *A Basic Stochastic Network Calculus*, ACM SIGCOMM Computer Communcation Review, 36 (2006), pp. 123–134.

[86] Y. JIANG AND Y. LIU, *Stochastic Network Calculus*, vol. 1, Springer, 2008.

[87] A. JOSSEN, V. SPÄTH, H. DÖRING, AND J. GARCHE, *Reliable Battery Operation - A Challenge for the Battery Management System*, Journal of Power Sources, 84 (1999), pp. 283–286.

[88] M. KAUER, S. NARANAYASWAMY, S. STEINHORST, M. LUKASIEWYCZ, S. CHAKRABORTY, AND L. HEDRICH, *Modular System-Level Architecture for Concurrent Cell Balancing*, in Proc. of Design Automation Conference (DAC), 2013.

[89] M. KAUER, S. NARAYANASWAMY, M. LUKASIEWYCZ, S. STEINHORST, AND S. CHAKRABORTY, *Inductor Optimization for Active Cell Balancing Using Geometric Programming*, in Proc. of Design, Automation and Test in Europe (DATE), 2015.

[90] M. KAUER, D. SOUDBAKHSH, D. GOSWAMI, S. CHAKRABORTY, AND A. M. ANNASWAMY, *Fault-tolerant Control Synthesis and Verification of Distributed Embedded Systems*, in Proc. of Design, Automation and Test in Europe (DATE), 2014.

[91] M. KAUER, S. STEINHORST, D. GOSWAMI, R. SCHNEIDER, M. LUKASIEWYCZ, AND S. CHAKRABORTY, *Formal Verification of Distributed Controllers using Time-Stamped Event Count Automata*, in Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC), 2013.

[92] M. KAUER, S. STEINHORST, R. SCHNEIDER, M. LUKASIEWYCZ, AND S. CHAKRABORTY, *Automata-Theoretic Modeling of Fixed-Priority Non-Preemptive Scheduling for Formal Timing Verification*, in Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC), 2014.

[93] H. K. KHALIL, *Nonlinear Systems*, Prentice Hall, 3 ed., 2001.

[94] H. KIM AND K. G. SHIN, *Dependable, Efficient, Scalable Architecture for Management of Large-Scale Batteries*, in Proc. of International Conference on Cyber-Physical Systems (ICCPS), 2010.

[95] I.-S. KIM, *A Technique for Estimating the State of Health of Lithium Batteries Through a Dual-Sliding-Mode Observer*, IEEE Transactions on Power Electronics, 25 (2010), pp. 1013–1022.

[96] R. KLEIN, N. CHATURVEDI, J. CHRISTENSEN, J. AHMED, R. FINDEISEN, AND A. KOJIC, *Electrochemical Model Based Observer Design for a Lithium-Ion Battery*, IEEE Transactions on Control Systems Technology, 21 (2013), pp. 289–301.

[97] H. KOPETZ AND G. BAUER, *The Time-Triggered Architecture*, Proceedings of the IEEE, 91 (2003), pp. 112–126.

[98] R. KORIES AND H. SCHMIDT-WALTER, *Electrical Engineering: A Pocket Reference*, Springer, 2003.

[99] J. KRAKORA AND Z. HANZALEK, *Timed Automata Approach to CAN Verification*, in Proc. of IFAC Symposium on Information Control Problems in Manufacturing (INCOM), 2005.

[100] R. KROEZE AND P. KREIN, *Electrical Battery Model for Use in Dynamic Electric Vehicle Simulations*, in Proc. of Power Electronics Specialists Conference (PESC), 2008.

[101] P. KUMAR, D. GOSWAMI, S. CHAKRABORTY, A. ANNASWAMY, K. LAMPKA, AND L. THIELE, *A Hybrid Approach to Cyber-Physical Systems Verification*, in Proc. of Design Automation Conference (DAC), 2012.

[102] N. KUTKUT AND D. DIVAN, *Dynamic Equalization Techniques for Series Battery Stacks*, in Proc. of Intl. Telecommunications Energy Conference (INTELEC), 1996.

[103] N. H. KUTKUT, *A Modular Nondissipative Current Diverter for EV Battery Charge Equalization*, in Proc. of Applied Power Electronics Conference and Exposition (APEC), 1998.

[104] K. LAMPKA, S. PERATHONER, AND L. THIELE, *Analytic Real-Time Analysis and Timed Automata: A Hybrid Methodology for the Performance Analysis of Embedded Real-Time Systems*, Design Automation for Embedded Systems, 14 (2010), pp. 193–227.

[105] M. LARABEL, *Linux Kernel at 19.5 Million Lines of Code, Continues Rising*, 2015. `http://www.phoronix.com/scan.php?page=news_item&px=Linux-19.5M-Stats`.

[106] K. G. LARSEN, P. PETTERSSON, AND W. YI, *Compositional and Symbolic Model-Checking of Real-Time Systems*, in Proc. of Real-Time Systems Symposium (RTSS), 1995.

[107] J. Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, Springer, 2001.

[108] E. A. Lee, *Cyber-Physical Systems - Are Computing Foundations Adequate?*, in Position Paper for NSF Workshop On Cyber-Physical Systems, 2006.

[109] E. A. Lee, *Cyber Physical Systems: Design Challenges*, in Proc. of Intl. Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008.

[110] J. Lee, O. Nam, and B. H. Cho, *Li-ion battery SOC estimation method based on the reduced order extended Kalman filtering*, Journal of Power Sources, 174 (2007), pp. 9–15.

[111] Y.-S. Lee and M.-W. Cheng, *Intelligent Control Battery Equalization for Series Connected Lithium-Ion Battery Strings*, IEEE Transactions on Industrial Electronics, 52 (2005), pp. 1297–1307.

[112] L. Li and M. Lemmon, *Weakly Coupled Event Triggered Output Feedback Control in Wireless Networked Control Systems*, in Proc. of Allerton Conf. on Communication, Control, and Computing, 2011.

[113] S. Li, C. C. Mi, and M. Zhang, *A High-Efficiency Active Battery-Balancing Circuit Using Multiwinding Transformer*, IEEE Transactions on Industry Applications, 49 (2013), pp. 198–207.

[114] D. Liberzon and A. S. Morse, *Basic Problems in Stability and Design of Switched Systems*, IEEE Control Systems, 19 (1999), pp. 59–70.

[115] B. Lincoln and A. Cervin, *Jitterbug: A Tool for Analysis of Real-Time Control Performance*, in Proc. of Conference on Decision and Control (CDC), 2002.

[116] Linear Technology, *LTC3300-1 - High Efficiency Bidirectional Multicell Battery Balancer*. `http://www.linear.com/product/LTC3300-1`.

[117] J. Liu, Z. Huang, J. Peng, and J. Wang, *Distributed Cooperative Voltage Equalization for Series-Connected Super-Capacitors*, in Proc. of American Control Conference (ACC), 2015.

[118] L. Ljung, *System Identification: Theory for the User*, Prentice Hall, (1987).

[119] S. Longo, G. Herrmann, and P. Barber, *Optimal Scheduling Methods for Time-Triggered Networked Control*, in Proc. of Intl. Conf. on Systems Engineering (ICSEng), 2009.

[120] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, *A review on the key issues for lithium-ion battery management in electric vehicles*, Journal of Power Sources, 226 (2013), pp. 272–288.

[121] X. Lu, W. Qian, and F. Z. Peng, *Modularized Buck-Boost + Cuk Converter for High Voltage Series Connected Battery Cells*, in Proc. of Applied Power Electronics Conference and Exposition (APEC), 2012.

[122] M. Lukasiewycz, M. Kauer, and S. Steinhorst, *Synthesis of Active Cell Balancing Architectures for Battery Packs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), (to appear).

[123] M. LUKASIEWYCZ, S. STEINHORST, AND S. NARAYANASWAMY, *Verification of Balancing Architectures for Modular Batteries*, in Proc. of Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014.

[124] S. K. MANDAL, P. S. BHOJWANI, S. P. MOHANTY, AND R. N. MAHAPATRA, *IntellBatt: Towards smarter battery design*, in Proc. of Design Automation Conference (DAC), 2008.

[125] A. MANENTI, A. ABBA, A. MERATI, S. M. SAVARESI, AND A. GERACI, *A New BMS Architecture Based on Cell Redundancy*, IEEE Transactions on Industrial Electronics, 58 (2011), pp. 4314–4322.

[126] O. MASON AND R. SHORTEN, *On common quadratic Lyapunov functions for stable discrete-time LTI systems*, IMA Journal of Applied Mathematics, 69 (2004), pp. 271–283.

[127] MAXIM INTEGRATED, *MAX11068 12-Channel, High-Voltage Sensor, Smart Data-Acquisition Interface*. `http://www.maximintegrated.com/en/products/power/battery-management/MAX11068.html`.

[128] M. MAZO AND P. TABUADA, *On Event-Triggered and Self-Triggered Control over Sensor/Actuator Networks*, in Proc. of Conference on Decision and Control (CDC), 2008.

[129] M. MAZO JR AND M. CAO, *Asynchronous Decentralized Event-triggered Control*, arXiv:1206.6648, (2012). `http://arxiv.org/abs/1206.6648`.

[130] M. MAZO JR, A. DAVITIAN, AND P. TABUADA, *Pessoa: A Tool for Embedded Controller Synthesis*, in Proc. of Computer Aided Verification (CAV), 2010.

[131] A. MOLIN AND S. HIRCHE, *Structural Characterization of Optimal Event-Based Controllers for Linear Stochastic Systems*, in Proc. of Conference on Decision and Control (CDC), 2010.

[132] A. MOLIN AND S. HIRCHE, *On the Optimality of Certainty Equivalence for Event-Triggered Control Systems*, IEEE Transactions on Automatic Control, 58 (2013), pp. 470–474.

[133] S. W. MOORE AND P. J. SCHNEIDER, *A Review of Cell Equalization Methods for Lithium Ion and Lithium Polymer Battery Systems*, SAE Technical Paper Series, (2001).

[134] M. MULANSKY AND K. AHNERT, *Odeint library*, Scholarpedia, 9 (2014), p. 32342. `http://www.scholarpedia.org/article/Odeint_library`.

[135] R. MURADORE, D. QUAGLIA, AND P. FIORINI, *Model Predictive Control over Delay-Based Differentiated Services Control Networks*, in Proc. of Design, Automation and Test in Europe (DATE), 2013.

[136] N. MURGOVSKI, L. JOHANNESSON, AND J. SJÖBERG, *Convex modeling of energy buffers in power control applications*, in Proc. of IFAC Workshop on Engine and Powertrain Control Simulation and Modeling, 2012.

[137] N. NAIK-DHUNGEL, *Energy Portfolio Standards and the Promotion of Combined Heat and Power*, U.S. Environmental Protection Agency (EPA), (2013).

[138] S. NARAYANASWAMY, S. STEINHORST, M. LUKASIEWYCZ, M. KAUER, AND S. CHAKRABORTY, *Optimal Dimensioning of Active Cell Balancing Architectures*, in Proc. of Design, Automation and Test in Europe (DATE), 2014.

[139] D. NEWCOMB, *The Next Big OS War Is in Your Dashboard*, 2012. http://www.wired.com/2012/12/automotive-os-war/.

[140] J. NEWMAN, K. E. THOMAS, H. HAFEZI, AND D. R. WHEELER, *Modeling of Lithium-Ion Batteries*, Journal of Power Sources, 119 (2003), pp. 838–843.

[141] ON SEMICONDUCTOR, *Power MOSFET NTLJS4114N*, 2012. http://www.onsemi.com/PowerSolutions/product.do?id=NTLJS4114N.

[142] A. OTTO, S. RZEPKA, T. MAGER, B. MICHEL, C. LANCIOTTI, T. GÜNTHER, AND O. KANOUN, *Battery Management Network for Fully Electrical Vehicles Featuring Smart Systems at Cell and Pack Level*, in Proc. of Advanced Microsystems for Automotive Applications (AMAA), 2012.

[143] C. PASCUAL AND P. KREIN, *Switched Capacitor System for Automatic Series Battery Equalization*, in Proc. of Applied Power Electronics Conference and Exposition (APEC), 1997.

[144] L. PHAN, S. CHAKRABORTY, AND P. THIAGARAJAN, *A Multi-mode Real-Time Calculus*, in Proc. of Real-Time Systems Symposium (RTSS), 2008.

[145] L. PHAN, S. CHAKRABORTY, P. THIAGARAJAN, AND L. THIELE, *Composing Functional and State-Based Performance Models for Analyzing Heterogeneous Real-Time Systems*, in Proc. of Real-Time Systems Symposium (RTSS), 2007.

[146] G. L. PLETT, *Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 1. Background*, Journal of Power Sources, 134 (2004), pp. 252–261.

[147] ——, *Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 2. Modeling and identification*, Journal of Power Sources, 134 (2004), pp. 262–276.

[148] ——, *Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs Part 3. State and parameter estimation*, Journal of Power Sources, 134 (2004), pp. 277–292.

[149] M. PREINDL, C. DANIELSON, AND F. BORRELLI, *Performance Evaluation of Battery Balancing Hardware*, in Proc. of European Control Conference (ECC), 2013.

[150] H. RAHIMI-EICHI, F. BARONTI, AND M.-Y. CHOW, *Online Adaptive Parameter Identification and State-of-Charge Coestimation for Lithium-Polymer Battery Cells*, IEEE Transactions on Industrial Electronics, 61 (2014), pp. 2053–2061.

[151] H. RAHIMI-EICHI, U. OJHA, F. BARONTI, AND M. CHOW, *Battery Management System: An Overview of Its Application in the Smart Grid and Electric Vehicles*, IEEE Industrial Electronics Magazine, 7 (2013), pp. 4–16.

[152] J.-P. RICHARD, *Time-Delay Systems: An Overview of Some Recent Advances and Open Problems*, Automatica, 39 (2003), pp. 1667–1694.

[153] P. RONG AND M. PEDRAM, *An Analytical Model for Predicting the Remaining Battery Capacity of Lithium-Ion Batteries*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 14 (2006), pp. 441–451.

[154] M. A. ROSCHER AND D. U. SAUER, *Dynamic electric behavior and open-circuit-voltage modeling of $LiFePO_4$-based lithium ion secondary batteries*, Journal of Power Sources, 196 (2011), pp. 331–336.

[155] S. SAMII, P. ELES, Z. PENG, P. TABUADA, AND A. CERVIN, *Dynamic Scheduling and Control-Quality Optimization of Self-Triggered Control Applications*, in Proc. of Real-Time Systems Symposium (RTSS), 2010.

[156] SAMSUNG SDI CO., LTD., *Introduction of INR18650-25R*, 2013. `http://www.datasheet-pdf.com/PDF/INR18650-25R-Datasheet-Samsung-839321`.

[157] J. SCHMALSTIEG, S. KÄBITZ, M. ECKER, AND D. U. SAUER, *A holistic aging model for $Li(NiMnCo)O_2$ based 18650 lithium-ion batteries*, Journal of Power Sources, 257 (2014), pp. 325–334.

[158] R. SCHNEIDER, D. GOSWAMI, S. ZAFAR, S. CHAKRABORTY, AND M. LUKASIEWYCZ, *Constraint-Driven Synthesis and Tool-Support for FlexRay-Based Automotive Control Systems*, in Proc. of Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011.

[159] B. SCHWEIGHOFER, K. RAAB, AND G. BRASSEUR, *Modeling of High Power Automotive Batteries by the Use of an Automated Test System*, IEEE Transactions on Instrumentation and Measurement, 52 (2003), pp. 1087–1091.

[160] H. SHIBATA, S. TANIGUCHI, K. ADACHI, K. YAMASAKI, G. ARIYOSHI, K. KAWATA, K. NISHIJIMA, AND K. HARADA, *Management of Serially-connected Battery System Using Multiple Switches*, in Proc. of Intl. Conference on Power Electronics and Drive Systems (PEDS), 2001.

[161] D. SHIN, M. PONCINO, E. MACII, AND N. CHANG, *A Statistical Model-Based Cell-to-Cell Variability Management of Li-ion Battery Pack*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34 (2015), pp. 252–265.

[162] J.-W. SHIN, G.-S. SEO, C.-Y. CHUN, AND B.-H. CHO, *Selective Flyback Balancing Circuit with Improved Balancing Speed for Series Connected Lithium-Ion Batteries*, in Proc. of Intl. Power Electronics Conference (IPEC), 2010.

[163] B. SINOPOLI, L. SCHENATO, M. FRANCESCHETTI, K. POOLLA, M. JORDAN, S. S. SASTRY, AND OTHERS, *Kalman Filtering with Intermittent Observations*, IEEE Transactions on Automatic Control, 49 (2004), pp. 1453–1464.

[164] D. SOUDBAKHSH AND A. M. ANNASWAMY, *Parallelized Model Predictive Control*, in Proc. of American Control Conference (ACC), 2013.

[165] D. SOUDBAKHSH, L. T. X. PHAN, O. SOKOLSKY, I. LEE, AND A. ANNASWAMY, *Co-design of Control and Platform with Dropped Signals*, in Proc. of International Conference on Cyber-Physical Systems (ICCPS), 2013.

[166] S. STEINHORST, M. KAUER, A. MEEUW, S. NARAYANASWAMY, M. LUKASIEWYCZ, AND S. CHAKRABORTY, *Cyber-Physical Co-Simulation Framework for Smart Cells in Scalable Battery Packs*, ACM Transactions on Design Automation of Electronic Systems (TODAES), (to appear).

[167] S. STEINHORST, M. LUKASIEWYCZ, S. NARAYANASWAMY, M. KAUER, AND S. CHAKRABORTY, *Smart Cells for Embedded Battery Management*, in Proc. of Intl. Conf. on Cyber-Physical Systems, Networks, and Applications (CPSNA), 2014.

[168] R. F. STENGEL, *Optimal Control and Estimation*, Courier Corporation, 2012.

[169] T. STUART, F. FANG, X. WANG, C. ASHTIANI, AND A. PESARAN, *A Modular Battery Management System for HEVs*, SAE Technical Paper, (2002).

[170] T. STUART AND W. ZHU, *Fast Equalization for Large Lithium Ion Batteries*, IEEE Aerospace and Electronic Systems Magazine, 24 (2009), pp. 27–31.

[171] Y. S. SUH, *Stability and Stabilization of Nonuniform Sampling Systems*, Automatica, 44 (2008), pp. 3222–3226.

[172] P. TABUADA, *Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks*, IEEE Transactions on Automatic Control, 52 (2007), pp. 1680 –1685.

[173] TEXAS INSTRUMENTS, *bq76PL102 - PowerLAN Dual-Cell Li-Ion Battery Monitor with Power-Pump Cell Balancing*. `http://www.ti.com/product/BQ76PL102`.

[174] ——, *BQ77PL900 5-10 cell Li-ion Battery protection & AFE*. `http://www.ti.com/product/BQ77PL900`.

[175] THE EUROPEAN COMMISSION, *Energy 2020*, European Union, (2011).

[176] L. THIELE, S. CHAKRABORTY, M. GRIES, AND S. KÜNZLI, *A Framework for Evaluating Design Tradeoffs in Packet Processing Architectures*, in Proc. of Design Automation Conference (DAC), 2002.

[177] L. THIELE, S. CHAKRABORTY, M. GRIES, A. MAXIAGUINE, AND J. GREUTERT, *Embedded Software in Network Processors - Models and Algorithms*, in Proc. of International Conference on Embedded Software (EMSOFT), 2001.

[178] L. THIELE, S. CHAKRABORTY, AND M. NAEDELE, *Real-Time Calculus for Scheduling Hard Real-Time Systems*, in Proc. of Intl. Symposium on Circuits and Systems (ISCAS), 2000.

[179] C. VAN LOAN, *Computing Integrals Involving the Matrix Exponential*, IEEE Transactions on Automatic Control, 23 (1978), pp. 395–404.

[180] H. VOIT, A. ANNASWAMY, R. SCHNEIDER, D. GOSWAMI, AND S. CHAKRABORTY, *Adaptive Switching Controllers for Systems with Hybrid Communication Protocols*, in Proc. of American Control Conference (ACC), 2012.

[181] W. WAAG, C. FLEISCHER, AND D. U. SAUER, *Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles*, Journal of Power Sources, 258 (2014), pp. 321–339.

[182] G. C. WALSH, H. YE, AND L. G. BUSHNELL, *Stability Analysis of Networked Control Systems*, IEEE Transactions on Control Systems Technology, 10 (2002), pp. 438–446.

[183] E. WANDELER, *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*, PhD thesis, Swiss Federal Institute of Technology Zurich, 2006.

[184] E. WANDELER AND L. THIELE, *Real-Time Calculus (RTC) Toolbox*, 2006. `http://www.mpa.ethz.ch/Rtctoolbox`.

[185] G. WEISS AND R. ALUR, *Automata Based Interfaces for Control and Scheduling*, in Proc. of Hybrid Systems: Computation and Control (HSCC), 2007.

[186] S. WEN, *Cell balancing buys extra run time and battery life*, Analog Applications Journal, (2009), pp. 14–18.

[187] WOLFRAM RESEARCH, INC., *Mathematica 9.0*, 2012.

[188] T. WONGPIROMSARN, U. TOPCU, N. OZAY, H. XU, AND R. M. MURRAY, *TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning*, in Proc. of Hybrid Systems: Computation and Control (HSCC), 2011.

[189] Y. XIONG, S. SUN, H. JIA, P. SHEA, AND Z. J. SHEN, *New Physical Insights on Power MOSFET Switching Losses*, IEEE Transactions on Power Electronics, 24 (2009), pp. 525–531.

[190] Y. XU AND J. HESPANHA, *Optimal Communication Logics in Networked Control Systems*, in Proc. of Conference on Decision and Control (CDC), 2004.

[191] Y. XU AND J. P. HESPANHA, *Estimation under uncontrolled and controlled communications in networked control systems*, in Proc. of European Control Conference (ECC), 2005.

[192] B. YANN LIAW, G. NAGASUBRAMANIAN, R. G. JUNGST, AND D. H. DOUGHTY, *Modeling of lithium ion cells – A simple equivalent-circuit model approach*, Solid State Ionics, 175 (2004), pp. 835–839.

[193] W. YE, J. HEIDEMANN, AND D. ESTRIN, *An Energy-Efficient MAC Protocol for Wireless Sensor Networks*, in Proc. of Conf. of the IEEE Computer and Communications Societies (INFOCOM), 2002.

[194] M. YU, L. WANG, G. XIE, AND T. CHU, *Stabilization of Networked Control Systems with Data Packet Dropout via Switched System Approach*, in Proc. of Intl. Symposium on Circuits and Systems (ISCAS), 2004.

[195] F. ZHANG, K. SZWAYKOWSKA, W. WOLF, AND V. MOONEY, *Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems*, in Proc. of Real-Time Systems Symposium (RTSS), 2008.

[196] H. ZHANG, G. DUAN, AND L. XIE, *Linear quadratic regulation for linear time-varying systems with multiple input delays*, Automatica, 42 (2006), pp. 1465–1476.

[197] W. ZHANG, M. BRANICKY, AND S. PHILLIPS, *Stability of Networked Control Systems*, IEEE Control Systems Magazine, 21 (2001), pp. 84–99.

List of Tables

List of Figures

List of Definitions & Theorems

Abbreviations

ACB Active Cell Balancing

API Application Programming Interface

BDD Binary Decision Diagram

BMS Battery Management System

CAN Controller Area Network

CAS Computer Algebra System

CPN Colored Petri Net

CPS Cyber-physical System

CPU Central Processing Unit

CQLF Common Quadratic Lyapunov Function

CSP Constraint Satisfaction Problem

DCC Drop Compensation Control

DSE Design Space Exploration

DP Dynamic Programming

ECA Event Count Automaton

ECU Electronic Control Unit

EES Electrical Energy Storage

EV Electric Vehicle

FPNS Fixed-Priority Non-preemptive Scheduling

GP Geometric Programming

GUI Graphical User Interface

IC Integrated Circuit

Li-Ion Lithium-Ion

LMI Linear Matrix Inequality

LQG Linear Quadratic Gaussian

LP Linear Programming

LTI Linear Time-invariant

LTL Linear Temporal Logic

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

MPC Model Predictive Control

NBA Non-determistic Büchi Automaton

NC Network Calculus

NCS Networked Control System

ODE Ordinary Differential Equation

OCV Open Circuit Voltage

PCB Printed Circuit Board

PDE Partial Differential Equation

PE Processing Element

PZOH Periodic Zero-Order Hold

QoC Quality of Control

RC Resistor-Capacitor

RTC Real-time Calculus

SAT Boolean Satisfiability

SoC State of Charge

TA Timed Automaton

TDMA Time Division Multiple Access

Nomenclature

Symbol	Units	Description
$\mathbf{0}$	$\mathbb{R}^{\star\times\star}$	Zero matrix: $m_{i,j} = 0 \quad \forall i, j$
$\mathbf{1}$	$\mathbb{R}^{\star\times\star}$	Identity matrix $\mathrm{diag}(1, \ldots, 1)$
$(\cdot)'$		Transposition operator
$(\cdot)^+$		Temporal next operator
A	$\mathbb{R}^{n\times n}$	System matrix
B	$\mathbb{R}^{n\times p}$	Input matrix
C	$\mathbb{R}^{q\times n}$	Output matrix
C_{OSS}	pF	Transistor output capacitance
c		Battery cell
d	ms	Deadline (fixed delay)
E_{sw}	$\mathrm{J} = \mathrm{V\,A\,s}$	Switching energy
h	ms	Sampling period
$i(t)$	A	Current
I	A	Peak inductor current
I_b	A	Break inductor current (Fig. 5.10)
k	h	Time steps (in periods)
L	H	Inductance
ℓ	$\mathbb{R}$	Energy loss ratio (balancing)
Q	A s	Charge
q	A s	Transferred charge increment
R	Ω	Resistance
R_C	Ω	Cell resistance
R_L	Ω	Inductor resistance
R_M	Ω	Transistor (MOSFET) resistance
$\cdot_r$		related to receiving cell
T	µs	Switching period (balancing)
t	s	Simulation time
$\cdot_t$		related to transmitting cell
τ	ms	Variable (actual) delay
τ_d	ns	Upwards transistor switching delay
τ_u	ns	Downwards transistor switching delay
$u(t)$, $u[k]$	$\mathbb{R}^p$	Input vector
V_d	V	Diode voltage drop
V_{ds}	V	Transistor drain-source voltage

ϕ		Switching phase (balancing)
$x(t)$, $x[k]$	$\mathbb{R}^n$	State vector
$y(t)$, $y[k]$	$\mathbb{R}^q$	Output vector
$z(t)$	$\%$	State of Charge (SoC) [Differences measured in pp or bp]

Index