TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Realzeit-Computersysteme

Hybrid Optimization Techniques for Multi-Domain Coupling in Cyber-Physical Systems Design

Debayan Roy

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. habil. Gerhard Rigoll
Prüfer der Dissertation:	1. Prof. Dr. sc. Samarjit Chakraborty, University of North Carolina at Chapel Hill, North Carolina, USA 2. Prof. Dr. Andreas Herkersdorf

Die Dissertation wurde am 22.04.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 30.10.2020 angenommen.

Abstract

In a cyber-physical system (CPS), the physical process is controlled by a software algorithm running on a hardware platform, where there is a *tight coupling* between the physical behavior of the system, the hardware, and the software. These systems are often *safety-critical* and have stringent performance requirements (e.g., faster stabilization after a disturbance). Many of them are also *cost-sensitive* and need to be implemented using limited platform resources (e.g., less number of communication slots or lower energy dissipation). However, the comprehensive design of a CPS requires the integration of *heterogeneous models* coming from different engineering domains, e.g., mechanical processes, electrical and electronic devices, control algorithms, and software implementations. Therefore, the design of high-performance yet resource-efficient CPSs leads to *challenging optimization problems*.

Challenges towards multi-domain coupling in CPSs design: There are several challenges to the comprehensive modeling and design of CPSs. First, it is not trivial to represent heterogeneous models in a *unified* framework. For example, a physical system evolves continuously according to a set of differential equations while the corresponding controller might be dispatched periodically and for each dispatch, it contests for the resource based on a certain scheduling policy that can be represented using a finite-state machine. Note that the theory of hybrid systems is often applied to tackle such a heterogeneity, however, such an approach often faces scalability issues. Second, with increasing *size* and *complexity* of industrial CPSs (like modern cars), the integrated design space grows exponentially, and therefore, the design problem can easily become intractable. Third, the requirements from different design domains might *conflict* with each other, e.g., improving control performance is often possible at the cost of more resources. Lastly, CPSs design typically involves a *disjoint set of tools* offered by different suppliers. It is non-trivial to develop an automated toolchain offering state-of-the-art design techniques while seamlessly interfacing the existing tools and preserving their advantages.

This thesis outlines our efforts in handling the aforementioned challenges using customized hybrid optimization techniques and novel design flows. We take automotive systems for our case study because they represent safety-critical, resource-constrained, and distributed CPSs.

Multi-objective co-optimization for distributed CPSs: Traditionally, in a CPS, control and platform designs are carried out in their isolated design trajectories and then integrated. This leads to an error-prone design or a long debugging and integration phase. In this context, we study a specific automotive setting where multiple distributed control applications share a FlexRay communication bus and propose a *correct-by-construction co-design approach* that simultaneously synthesizes control and platform parameters. In the emerging area of control-platform co-design, for the first time, we consider to *co-optimize* control performance and resource usage, thereby, obtaining a *Pareto front* where each Pareto point is a valid design configu-

ration representing a trade-off between the two design objectives. While such a co-optimization problem is often intractable, we explore the large design space using a tailor-made *two-stage optimization* technique. In the first stage, for each application, we design *prospective* optimal controllers using different amounts of resources considering constraints imposed by the FlexRay protocol. In the second stage, we employ a nested *three-layer* optimization technique to determine the Pareto front. In the outermost layer, we iterate through different possible values of resource usage. In the second layer, we solve a *mixed-integer linear programming (MILP)* problem to determine an optimal set of controllers that uses the given amount of resources and maximizes the system performance. In the innermost layer, we try to find a feasible schedule for the selected set of controllers by solving an *integer linear programming (ILP)* problem. Results show that for a relatively small case study comprising 5 applications mapped on to 3 electronic control units (ECUs), the average control performance can be improved by $41.14\,\%$ of the required value at the cost of $27.5\,\%$ more communication resource, i.e., there is a significant trade-off opportunity. Our approach also scales to a bus cluster comprising 24 applications.

Tool integration for automated synthesis and implementation of distributed CPSs: While we have developed a state-of-the-art co-design approach for FlexRay-based distributed automotive CPSs, the lack of integrated industrial tools prevents it to be evaluated in industrial settings. In effect, this leads to a *gap* in the state-of-the-art and the state-of-practice. Towards bridging this gap, we have developed a *toolchain* by integrating the proposed co-optimization approach into commercially available tools. This toolchain uses MATLAB/Simulink for modeling control systems and SIMTOOLS/SIMTARGET (offered by Elektrobit) for modeling the distributed embedded platform. Besides exploiting the functionalities of existing tools, we also offer tools that automate specification extraction, control-platform co-design, and parameter configuration. Therefore, the proposed toolchain enables *design automation* and therefore, allows convenient design and implementation of FlexRay-based systems. Corresponding to the proposed toolchain, we also introduce a customized methodology to develop control software using a correct-by-construction technique.

Tighter dimensioning of multi-resource CPSs with control performance guarantees: We consider a multi-resource CPS setup comprising high- and low-quality platform resources. While a high-quality resource offers timing guarantees, and hence, a higher control performance, they are significantly expensive. In cost-sensitive CPS domains like automotive, it is desirable to optimize the usage of such expensive resources. Towards this goal, we study a control scheme that comprises *two modes* using high-quality and low-quality resources respectively. We propose *cost-efficient yet safe static and dynamic scheduling strategies* to allocate high-quality resources to the bimodal controllers. For each application, we first study the switching control dynamics to derive the minimum requirement for high-quality resources based on control specification. Subsequently, we formulate a Satisfiability Modulo Theories (SMT) problem to statically allocate the required resources to the applications while maximizing the *extensibility* of the schedule. Furthermore, for each application, we derive: (i) the maximum time for which the controller can wait for high-quality resources without violating the performance constraint, and (ii) for certain wait time, the minimum and the maximum time for which the controller can use the high-quality resources continuously. Using these timing information, we devise a safe switching scheme for the controllers and a scheduling policy based on dynamically computed priorities of the applications. The resource dimensioning problem for the dynamic scheduling is

solved in two nested layers where the outer layer is a *first-fit heuristic* while in the inner layer we formally verify a network of *timed-automata* representing the applications and the scheduler. Results suggest that resource dimensioning can be significantly *tightened* (i.e., 50% in certain cases) using the proposed scheduling strategies compared to existing approaches.

Energy- and time-optimal active cell balancing in battery packs: *Electric vehicles (EVs)* are becoming the mainstream in the automotive industry. In EVs, high-power battery packs, comprising several hundreds of cells in series and parallel, act as the main energy source. Safety and performance of such a pack are ensured by a hardware/software system, i.e., the *battery management system (BMS)*. Cell balancing is a crucial task of BMSs that maximizes the usable capacity of the pack. In active cell balancing, the charge is transferred between series-connected cells such that the charge levels of all cells become equal. The problem of cell balancing has been primarily studied in the power-electronics community where low-power charge transfer circuits have been proposed. However, control algorithms operating these circuits have relied only on heuristics. No results on optimal cell balancing with respect to energy dissipation and balancing time respectively were known prior to our work. We, for the first time, formulate the cell balancing problem from a real-time CPS perspective and propose control algorithms that minimize energy dissipation and balancing time respectively. We show that holistic optimization is possible by integrated modeling of the cell chemistry, charge transfer process, charge equalization, and the constraints imposed by the charge transfer circuit and by the BMS architecture. We *partition the cell balancing problem into two stages without losing any optimality guarantee*. In the first stage, we formulate *MILP* problems to determine the time- and the energy-optimal sets of charge transfers respectively that will realize charge equalization. In the second stage, we formulate a *minimum vertex coloring (MVC)* problem to schedule the charge transfer sets obtained in the first stage. Using the proposed optimization, balancing time can be improved from $11.04\,\mathrm{h}$ to $4.82\,\mathrm{h}$ and energy dissipation can be reduced from $211\,\mathrm{Wh}$ to $133\,\mathrm{Wh}$ for certain balancing scenarios. Our proposed technique has real benefits in *improving the charging time* and *extending the driving range* of EVs.

Summary: This thesis shows that *holistic optimization* is possible in the design of CPSs when models from different engineering domains are considered in an *integrated* framework. Towards the optimal design of CPSs, it is important to determine the *interplay* between different design domains. Exploiting the interplay and by studying the characteristics of the design problems, customized optimization techniques are developed towards *efficient and scalable design space exploration (DSE)* for CPSs.

Kurzfassung

In einem cyber-physischen System (CPS) wird der physikalische Prozess durch einen Software-Algorithmus gesteuert, der auf einer Hardware-Plattform läuft. Es existiert eine enge Kopplung zwischen dem physikalischen Verhalten des Systems, der Hardware und der Software. Derartige Systeme sind oft sicherheitskritisch und haben strenge Leistungsanforderungen (z.B. schnelle Stabilisierung nach einer Störung). Häufig sind solch Systeme auch kosten-kritisch und müssen mit begrenzten Plattform Ressourcen auskommen (z.B. eine geringe Zahl Kommunikationsslots oder geringe Verlustenergie). Der umfassende Entwurf eines CPS erfordert jedoch die Integration heterogener Modelle, die aus verschiedenen Engineering-domänen stammen, z.B. mechanische Prozesse, elektrische und elektronische Geräte, Regelungsalgorithmen und Software-Implementierungen. Daher führt der Entwurf von hochleistungsfähigen und dennoch ressourceneffizienten CPS zu anspruchsvollen Optimierungsproblemen.

Herausforderungen bei der Multi-Domain-Kopplung beim Entwurf von CPS: Die umfassende Modellierung und der Entwurf von CPS sind mit mehreren Herausforderungen verbunden. Erstens ist es nicht trivial, heterogene Modelle in einem einheitlichen Framework darzustellen. Beispielsweise entwickelt sich ein physikalisches System kontinuierlich gemäß eines Satzes von Differentialgleichungen, während der entsprechende Controller möglicherweise periodisch operiert, und bei jedem Einsatz konkurriert er um die Ressourcen je nach der eingesetzen Scheduling Policy, die mit einem endlichen Zustandsautomat dargestellt werden kann. Es ist zu beachten, dass häufig Hybridsystem-Theorie angewandt wird, um eine solche Heterogenität zu bewältigen, wobei ein solcher Ansatz jedoch oft mit Skalierbarkeitsproblemen einhergeht. Zweitens wächst mit zunehmender Größe und Komplexität industrieller CPS (wie sie bei modernen Fahrzeugen eingesetzt werden) der integrierte Designraum exponentiell an, so dass das Designproblem leicht unlösbar werden kann. Drittens könnten die Anforderungen aus verschiedenen Designbereichen miteinander in Konflikt geraten, z. B. ist die Verbesserung der Steuerungsleistung oft auf Kosten von mehr Ressourcen möglich. Schließlich beinhaltet der Entwurf von CPS in der Regel einen disjunkten Satz von Werkzeugen, die von verschiedenen Anbietern angeboten werden. Es ist nicht trivial, eine automatisierte Toolchain zu entwickeln, die modernste Entwurfstechniken bietet und gleichzeitig eine nahtlose Verbindung zu den bestehenden Werkzeugen herstellt und deren Vorteile bewahrt.

Diese Arbeit beschreibt unsere Bemühungen zur Bewältigung der oben genannten Herausforderungen unter Verwendung maßgeschneiderter hybrider Optimierungstechniken und neuartiger Design-Flows. Wir nehmen Automobilsysteme für unsere Fallstudie an, weil sie sicherheitskritische, ressourcenbeschränkte und verteilte CPS darstellen.

Pareto-Optimierung für verteilte CPS: Traditionell findet beim Design von CPS die Entwicklung von Regelung und Plattform getrennt statt. Erst am Ende werden beide Entwürfe integriert.

Dies führt zu einem fehleranfälligen Design oder zu einer langen Debugging- und Integrationsphase. In diesem Zusammenhang untersuchen wir eine spezifische Automobilumgebung, in der sich mehrere verteilte Steuerungsanwendungen einen FlexRay-Kommunikationsbus teilen, und schlagen einen Correct-by-Construction-Co-Design-Ansatz vor, bei dem Steuerungs- und Plattformparameter gleichzeitig synthetisiert werden. Im noch jungen Gebiets des Co-Designs von Steuerungsplattformen erwägen wir zum ersten Mal, die Steuerungsleistung und den Ressourcenverbrauch zu optimieren und dadurch eine Pareto-Front zu erhalten, bei der jeder Pareto-Punkt eine gültige Designkonfiguration darstellt, die einen Kompromiss zwischen den beiden Designzielen darstellt. Während ein solches Co-Optimierungsproblem oft unlösbar ist, untersuchen wir den großen Designraum mit einer maßgeschneiderten zweistufigen Optimierungstechnik. In der ersten Phase entwerfen wir für jede Anwendung potentielle optimale Controller unter Verwendung unterschiedlicher Ressourcenmengen und unter Berücksichtigung der durch das FlexRay-Protokoll auferlegten Einschränkungen. In der zweiten Stufe verwenden wir eine verschachtelte drei-stufige Optimierungstechnik zur Bestimmung der Pareto-Front. In der ersten Stufe iterieren wir durch verschiedene mögliche Werte der Ressourcennutzung. In der zweiten Stufe lösen wir ein MILP-Problem (Mixed-Integer Linear Programming), um einen optimalen Satz von Controllern zu bestimmen, der die gegebene Menge an Ressourcen nutzt und die Systemleistung maximiert. In der letzten Stufe versuchen wir, durch die Lösung eines ganzzahligen linearen Programmierungsproblems (ILP-Problems) eine realisierbaren Schedule für den ausgewählten Satz von Controllern zu finden. Die Ergebnisse zeigen, dass bei einer relativ kleinen Fallstudie mit 5 Anwendungen, die auf 3 elektronische Steuergeräte (ECUs) abgebildet werden, die durchschnittliche Regelleistung um 41,14% des erforderlichen Wertes verbessert werden kann, und das bei Kosten von 27,5% mehr Kommunikationsressourcen. D.h. es besteht ein erheblicher Spielraum für Kompromisslösungen. Unser Ansatz lässt sich auch auf ein Bus-Cluster mit 24 Anwendungen skalieren.

Tool-Integration für die automatisierte Synthese und Implementierung verteilter CPS: Wir haben einen hochmodernen Co-Design-Ansatz für FlexRay-basierte verteilte CPS in der Automobilindustrie entwickelt, jedoch verhindert das Fehlen integrierter industrieller Tools eine Evaluierung in industriellen Umgebungen. Tatsächlich führt dies zu einer Lücke im Stand der Technik und in der Anwendungspraxis. Um diese Lücke zu überbrücken, haben wir eine Toolchain entwickelt, indem wir den vorgeschlagenen Co-Optimierungsansatz in kommerziell verfügbare Werkzeuge integriert haben. Diese Toolchain verwendet MATLAB/Simulink für die Modellierung von Steuerungssystemen und SIMTOOLS/SIMTARGET (von Elektrobit) für die Modellierung der verteilten eingebetteten Plattformen. Neben der Nutzung der Funktionalitäten bestehender Werkzeuge bieten wir auch Werkzeuge an, die die Extraktion von Spezifikationen, das Co-Design von Steuerungsplattformen und die Parameterkonfiguration automatisieren. Daher ermöglicht die vorgeschlagene Werkzeugkette die Automatisierung des Designs und damit bequemen Entwurf und Implementierung von FlexRay-basierten Systemen. Entsprechend der vorgeschlagenen Toolchain führen wir auch eine angepasste Methodik zur Entwicklung von Steuerungssoftware mit einer Correct-by-Construction-Technik ein.

Engere Dimensionierung von Multi-Ressourcen-CPS mit Kontrollleistungsgarantien: Wir betrachten einen Multi-Ressourcen-CPS-Aufbau mit hochwertigen und minderwertigen Plattformressourcen. Eine qualitativ hochwertige Ressource bietet zwar Timing-Garantien und damit eine höhere Steuerungsleistung, ist aber erheblich teurer. In kostensensiblen CPS-

Anwendungsgebieten wie der Automobilindustrie ist es wünschenswert, den Einsatz solch teurer Ressourcen zu optimieren. Um dieses Ziel zu erreichen, untersuchen wir ein Kontrollschema, das zwei Modi umfasst, die qualitativ höherwertige bzw. qualitativ minderwertige Ressourcen nutzen. Und wir schlagen kosteneffiziente und dennoch sichere statische und dynamische Planungsstrategien vor, um den bi-modalen Steuerungseinheiten qualitativ hochwertige Ressourcen zuzuweisen. Für jede Anwendung untersuchen wir zunächst die Schalt-Regeldynamik, um auf der Grundlage der Regelspezifikation die Mindestanforderung an hochwertige Ressourcen abzuleiten. Anschließend formulieren wir ein Problem der Satisfiability Modulo Theories (SMT) um den Anwendungen die erforderlichen Ressourcen statisch zuzuweisen und gleichzeitig die Erweiterbarkeit des Schedulers zu maximieren. Darüber hinaus leiten wir für jede Anwendung folgendes ab: (i) die maximale Zeitspanne, während welcher der Controller auf qualitativ hochwertige Ressourcen warten kann ohne die Leistungsbeschränkung zu verletzen, und (ii) für bestimmte Wartezeiten die minimale und maximale Zeitspanne, während welcher der Controller die qualitativ hochwertigen Ressourcen kontinuierlich nutzen kann. Unter Verwendung dieser Zeitinformationen entwickeln wir ein sicheres Umschaltschema für die Controller und eine Planungsrichtlinie, die auf dynamisch berechneten Prioritäten der Anwendungen basiert. Das Ressourcendimensionierungsproblem für das dynamische Scheduling wird in zwei verschachtelten Schichten gelöst, wobei die äußere Schicht eine First-Fit-Heuristik ist, während wir in der inneren Schicht ein Netzwerk von timed automata, die die Anwendungen und den Scheduler repräsentieren, formal verifizieren. Die Ergebnisse deuten darauf hin, dass die Ressourcendimensionierung mit den vorgeschlagenen Scheduling Strategien im Vergleich zu bestehenden Ansätzen deutlich (d.h. in bestimmten Fällen um 50%) gestrafft werden kann.

Energie- und zeitoptimales aktives Cell Balancing in Akkupacks: Elektrofahrzeuge (Electric Vehicles, EVs) werden mehr und mehr zum Mainstream in der Automobilindustrie. In EVs dienen Hochleistungsbatteriepakete, die aus mehreren hundert Zellen bestehen, welche in Serie sowie parallel geschaltet sind, als Hauptenergiequelle. Sicherheit und Leistung eines solchen Packs werden durch ein Hardware/Software-System, d. h. das Batteriemanagementsystem (BMS), gewährleistet. Das Ausgleichen der Batteriezellen (cell balancing) ist eine entscheidende Aufgabe der BMS, die die nutzbare Kapazität des Packs maximiert. Beim aktiven Cell Balancing wird die Ladung zwischen in Reihe geschalteten Zellen so übertragen, dass die Ladungsniveaus aller Zellen ausgeglichen werden. Das Problem des Cell Balancings wurde vor allem in der Leistungselektronik untersucht, wo Ladungsübertragungsschaltungen mit geringer Leistung vorgeschlagen wurden. Die Steueralgorithmen, die diese Schaltkreise betreiben, haben sich jedoch nur auf Heuristiken gestützt. Vor unserer Arbeit waren keine Ergebnisse zum optimalen Cell Balancing in Bezug auf Energiedissipation bzw. Ausgleichsdauer bekannt. Zum ersten Mal formulieren wir das Cell Balancing Problem aus einer Echtzeit-CPS-Perspektive und schlagen Regelalgorithmen vor, die die Energiedissipation bzw. die Ausgleichsdauer minimieren. Wir zeigen, dass eine ganzheitliche Optimierung durch integrierte Modellierung der Zellchemie, des Ladungstransferprozesses, des Ladungsausgleichs und der durch die Ladungsübertragungsschaltung und die BMS-Architektur auferlegten Einschränkungen möglich ist. Wir teilen das Problem des Cell Balancing in zwei Phasen auf, ohne dabei die Garnatie auf Optimalität zu verlieren. In der ersten Stufe formulieren wir MILP-Probleme, um die zeit- bzw. energieoptimalen Mengen von Ladungstransfers zu bestimmen, die einen Ladungsausgleich realisieren. In der zweiten Stufe formulieren wir ein MVC-Problem (Minimum Vertex Coloring), um

die in der ersten Stufe erhaltenen Ladungstransfermengen zu planen. Mit der vorgeschlagenen Optimierung kann die Balancing Dauer von 11,04 h auf 4,82 h verbessert und die Energiedissipation für bestimmte Ausgleichsszenarien von 211Wh auf 133Wh reduziert werden. Die von uns vorgeschlagene Technik hat als Vorteile, dass sie sowohl die Ladezeit verbessert als auch die Reichweite von EVs erhöht.

Zusammenfassung: Diese Arbeit zeigt, dass eine ganzheitliche Optimierung bei der Gestaltung von CPS möglich ist, wenn Modelle aus verschiedenen Engineering-Domänen in einem integrierten Framework betrachtet werden. Für die optimale Entwicklung von CPS ist es wichtig, das Zusammenspiel zwischen verschiedenen Entwurfsbereichen zu bestimmen. Unter Ausnutzung des Zusammenspiels und durch Untersuchung der Charakteristika der Entwurfsprobleme werden maßgeschneiderte Optimierungstechniken zur effizienten und skalierbaren Design Space Exploration (DSE) für CPS entwickelt.

Acknowledgements

At the end of 4 years 7 months 19 days, finally I get an opportunity to reflect on my journey as a researcher so far. During this journey, there are several people who has supported me in different ways, and I would like to thank them all.

First and foremost, I am grateful to Samarjit Chakraborty for his constant guidance, support, and encouragement. He interviewed me for my master's admission at TU Munich, and then, co-incidentally, we met on the very first week I landed in Munich, and since then I have always looked up to him as my advisor. He offered me a HiWi position soon after and now, I am employed at the Institute of Real-Time Computer systems for more than 6 years. As a doctoral advisor, he always had excellent inputs in my research. His constant determination to solve challenging problems and do high-quality research has motivated me to publish in top-ranked conferences and journals. I am also grateful to him for having me introduced to academics around the world with whom I have had successful collaborations. Nevertheless, during these years, we have had several disagreements. I must admit that he has always understood my position and supported my decision. I apologize for the inconveniences caused to him.

Furthermore, I am thankful to Prof. Herkersdorf for agreeing to examine my thesis. Coming from a more traditional electrical engineering background, the courses he offered during my master's have helped me to develop a good background on processor architecture and scheduling, memory architecture, and hardware/software co-design. This has helped me a lot in my research. Also, a hearty thank to Prof. Rigoll for agreeing to head the examination committee. He always comes across as a very nice person.

Sincere thanks to the co-authors of the publications I have had. Licong Zhang has mentored me in my early days as a researcher. I have revisited control theory using Dip Goswami's teaching material and also had several enriching discussions with him. Wanli Chang and Swaminathan Narayanaswamy have been excellent to work with. I am grateful to Prof. Sanjoy Mitter to have spent so much time with me discussing research ideas, while also hosting me at MIT. I greatly appreciate Qi Zhu to have hosted me at the Northwestern University and I hope to follow up soon and write some papers from the collaboration. I would also like to mention Anuradha Annaswamy, Xiaobo Sharon Hu, Thomas Heurung, Michael Balszun, Alma Pröbstl, Anna Minaeva, Soumyajit Dey, and Sumana Ghosh for successful research collaborations.

The research for this doctoral thesis has been conducted in the Institute of Real-Time Computer System at TU Munich. Thanks to Irene Dippold, Renate Schwarz, Kathrin Schmöller, and Reinald Gfüllner for their help and support in conducting operations in the institute including teaching, organizing travels, claiming expenses, among others. A special thank to each and every colleague at RCS as I am sure I have received help at some point in time during these years. I would like to mention Alma Pröbstl with whom I have shared office. Besides research collaborations, she has been of great help especially with my poor German skills. Thanks to Philipp Kindt, Daniel Yunge, Nils Heitmann, and Martin Geier for several enriching discussions and to Nadja Heitmann for her help regarding administrative and several other important matters.

The superstar in this journey who has contributed to each and every achievement I have had until now without asking for a credit has been my wife, Mallika Dey. I cannot thank her enough for her constant support. Samarjit has never been a nagging supervisor (of course I am not complaining), however, she always kept me on my toes to produce results before submission deadlines. She always lent ears to my ideas and with her background in computer science, discussions with her have always been productive. In retrospect, I would add her as a co-author in a couple of my publications. The biggest thank goes to our tiny 4-month old daughter, Raima Roy, to have fought the bravest battle after her birth. She has been the greatest encouragement towards writing and submitting my thesis in the past few months.

My sincere gratitude to my father, Aloke Kumar Roy, to have encouraged me in the best way he could, to pursue higher studies in Germany. I hope my mother, Debahuti Roy, is blessing me from wherever she is now, and I believe that she is making unexpected things happen for me by coaxing the almighty. I have a great family who always keep me amused. My brother, Debalok Roy, my cousins and my sisters-in-law never fail to entertain me whenever I am down.

I do not expect anyone to read this far, however, if you have and have not found your name, I apologize and would like to thank you as I am sure you are my well wisher.

Contents

1 Introduction

The term “cyber-physical system” was coined by Helen Gill in 2006 at the National Science Foundation in the United States [1]. In a cyber-physical system (CPS), the physical process is controlled by a hardware/software (cyber) system. That is, the controller, monitoring the physical process and taking suitable control actions, is implemented as software components on the hardware platform. These software components (or tasks) are executed using computation and memory resources. In a distributed setting, these tasks are mapped on different processing units and data transmission between them uses communication resources.

Traditionally, control engineers design controllers taking into consideration the dynamical models of the controlled plants. On the other hand, computer engineers are only focussed on developing algorithms that enable efficient usage of hardware resources (i.e., computation, memory, and communication resources). However, for a CPS, there is a strong interplay between the control algorithm, the physical dynamics of the controlled plant and the hardware resources. It is very important to consider this interplay while designing the CPSs [2].

Due to the aforementioned interplay, the physical behavior of a CPS depends not only on the plant dynamics and the control law, but also on how the controller is implemented on the hardware platform, i.e., the types of hardware resources (i.e., time- or event-triggered resources) used and how the resources are allocated (i.e., the schedule) to the controller. In other words, the choice of an optimal controller depends on the dynamical model of the controlled plant and the timings of the control software [3]. Thus, for designing the controller, the relevant software timing information are as follows: (i) What kind of sampling periods are possible? (ii) What is the delay between sampling and actuation? (iii) How much is the sampling jitter? (iv) How much is the output jitter?

Note that the above timing details are typically obtained based on the platform implementation of the controller and they might not be known apriori during the controller design. In that case, certain assumptions are made on the controller implementation. These assumptions must be then considered as specification while implementing the controller [4]. In the event when a timing assumption made during controller design could not be realized in the implementation, the control guarantees provided during the controller design might not be preserved in the implementation and safety can be jeopardized [5]. For example, the controller optimized based on a sampling period of $5\,\mathrm{ms}$, when implemented using a control task that runs every $10\,\mathrm{ms}$, may jeopardize the stability of the closed-loop system. In the same vein, an optimal controller, designed considering a sampling period of $10\,\mathrm{ms}$, might become sub-optimal when implemented by a control task scheduled every $5\,\mathrm{ms}$.

CPSs are common in domains such as automotive, avionics, industrial automation, energy, health care, and defense. Some examples of CPSs are as follows:

- *Adaptive Cruise Control Systems*: Modern cars are equipped with the adaptive cruise control feature, i.e., the speed of the car is automatically controlled to maintain a safe distance from the car in front.

- *Cyber-Physical Traffic Control Systems*: In modern transportation networks, signalized intersections are controlled based on real-time traffic information (i.e., queue lengths at incoming and outgoing roads) and they have proved to be more efficient in reducing traffic delays.

- *Artificial Pancreas Systems*: In patients with type 1 diabetes, artificial pancreas systems track blood glucose level continuously and automatically delivers hormone insulin when necessary.

- *Heating, Ventilation and Air-Conditioning (HVAC) Systems*: In smart homes and modern cars, HVAC systems are used to monitor the ambience of the interior and provide thermal comfort and ensure good indoor air quality.

- *Smart Grids*: In smart grids, advanced monitoring and forecasting is performed to deliver energy reliably and with high operation efficiency for generators and distributors.

- *Swarm Robotics*: In swarm robotics, a group of co-ordinated robots are released in a physical environment to gather information and take necessary control actions. For example, in smart agriculture, a number of unmanned aerial vehicles can be deployed in large farmlands to gather real-time information about plant health and soil quality and these information can then be used to automatically water the plants, and provide fertilizers and pesticides. Besides, swarm robotics has found its application in the field of health care, defense, smart manufacturing, among others.

Chapter organization: The rest of this chapter is organized into five sections. Section 1.1 briefly describes the automotive CPS setup considered in this thesis. In Section 1.2, we first motivate that a holistic approach is necessary integrating models from different design domains towards safe and efficient design of CPSs. We further study the challenges towards multi-domain coupling in CPSs. Section 1.3 outlines the evolution of design paradigms for CPSs. In Section 1.4, we mention the scientific contributions of this thesis and provide the structure of this thesis. Section 1.5 provides a list of publications that this thesis and other associated works led to.

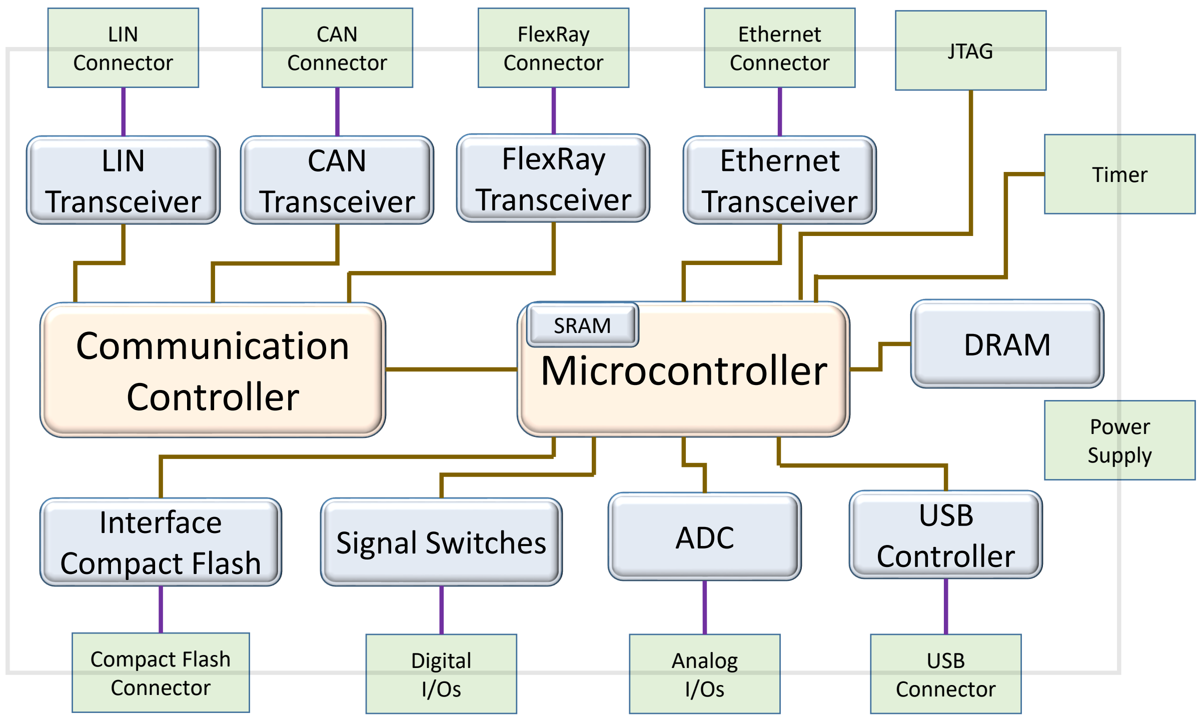


Figure 1.1: A schematic of an automotive electronic control unit (ECU).

1.1 Automotive Cyber-Physical Systems

This thesis mainly studies automotive systems. Modern automotive electrical and electronic (E/E) architectures are highly complex and heterogeneous with up to 100 electronic control units (ECUs) connected over a communication network [6, 7]. The communication network comprises various communication buses like FlexRay, CAN, and more recently also automotive Ethernet. Several hundred million lines of software code are running on an automotive E/E platform [8]. In these systems, software components typically implement control functions that determine the physical dynamics of the car. The control applications come from various domains [9] – like basic automotive functionality (brake control, engine control), advanced driver assistance systems (cruise control, lane control), and also comfort features (like vibration control). In electric vehicles (EVs), the battery pack is the main source of energy. It powers the car, thereby, determining the maximum accerleration and the driving range of the car. These high-power battery packs are typically provided with a hardware/software system called the battery management system (BMS) [10]. The main functions of a BMS include monitoring the state of the battery pack, maintaining its safe operation and ensuring the desired usable capacity of the pack [11]. In this section, different components of automotive CPSs will be briefly discussed.

1.1.1 Electronic Control Units

Typically, as shown in Figure 1.1, an ECU in a modern car comprises a core (i.e., a micro-controller), memory resources (i.e., SRAM and Flash), inputs (i.e., supply voltage, digital outputs, analog inputs), outputs (i.e., relay drivers, H bridge drivers, logic outputs, injector drivers), communication controller (i.e., an FPGA or a micro-controller), and network interfaces (i.e., transceivers) [7]. Thus, software gets information about the physical system from the sensors connected to the inputs of the ECU. Based on the inputs, software runs using the computation and memory resources in the ECU. Control decisions taken by the software are then applied to the physical system using the outputs of the ECU. In case of a distributed software implementation, software components communicate by sending data using the communication controllers and transceivers (on sending and receiving side respectively) over the communication bus.

Communication controllers perform communication based on the corresponding communication protocol. For example, E-Ray IP module supports FlexRay v2.1 [12]. Communication data, are therefore, first sent to the communication controller, where they are packetized into communication frames as allowed in the protocol. Then, these frame are sent over the communication bus based on their respective schedules and the scheduling policy of the protocol. For example, in case of FlexRay protocol, a frame will be sent only when the current slot id matches the slot id assigned to the frame. On the receiving ECU, the communication controller monitors network activities. When there is a frame that is sent to the host micro-controller, it receives the frame, depacketizes the frame to get the data and transmit the corresponding data to the host micro-controller.

In the automotive industry, there is a large variation in the capabilities of the available ECUs. For example, a low-end ECU might only have a 8 bit processor with 8 MHz, 4 kB flash memory, and 256 byte RAM while a top-end automotive ECU might have a 32 bit multi-core processor with 200 MHz, 6 MB flash memory, and 384 kB RAM[1].

Typically, the software running on an ECU might be composed of several functions. For example, a combustion engine controller implements various independent functions, such as the adjustment of the fuel injection while monitoring the rotational speed of the turbo charger [13]. In the current state-of-practice, these functions do not run on the bare metal. Automotive ECUs run a real-time operating system (RTOS) that manages the system resources. Thus, the software functions mapped on to an ECU are scheduled based on the RTOS employed.

OSEK/VDX: In 1993, German automotive consortium (inlcuding BMW, Robert Bosch GmbH, DaimlerChrysler, Opel, Siemens and Volkswagen Group) and the University of Karlsruhe founded OSEK (Offene Systeme und deren Schnittstellen für Elektronik in Kraftfahrzeugen) towards standardizing the software architecture [14]. In 1994, French car manufacturers Renault and PSA Peugeot Citroën merged their project Vehicle Distributed eXecutive (VDX) with OSEK, and formulated OSEK/VDX [15]. This group specified a standard for automotive RTOS (*Offene Systeme und deren Schnittstellen für Elektronik in Kraftfahrzeugen (OSEK)-OS*) [16] that has been widely deployed in automotive ECUs. OSEK-OS provides standards for task management, synchronization, interrupt management, alarms, intra-processor message handling, and error treatment. The earlier specification supported the event-triggered scheduling

[1]Data for Freescale automotive micro-controllers *S08QD* and *MPC5676R*, taken from http://cache.freescale.com/files/microcontrollers/doc/roadmap/BRAUTOPRDCTMAP.pdf.

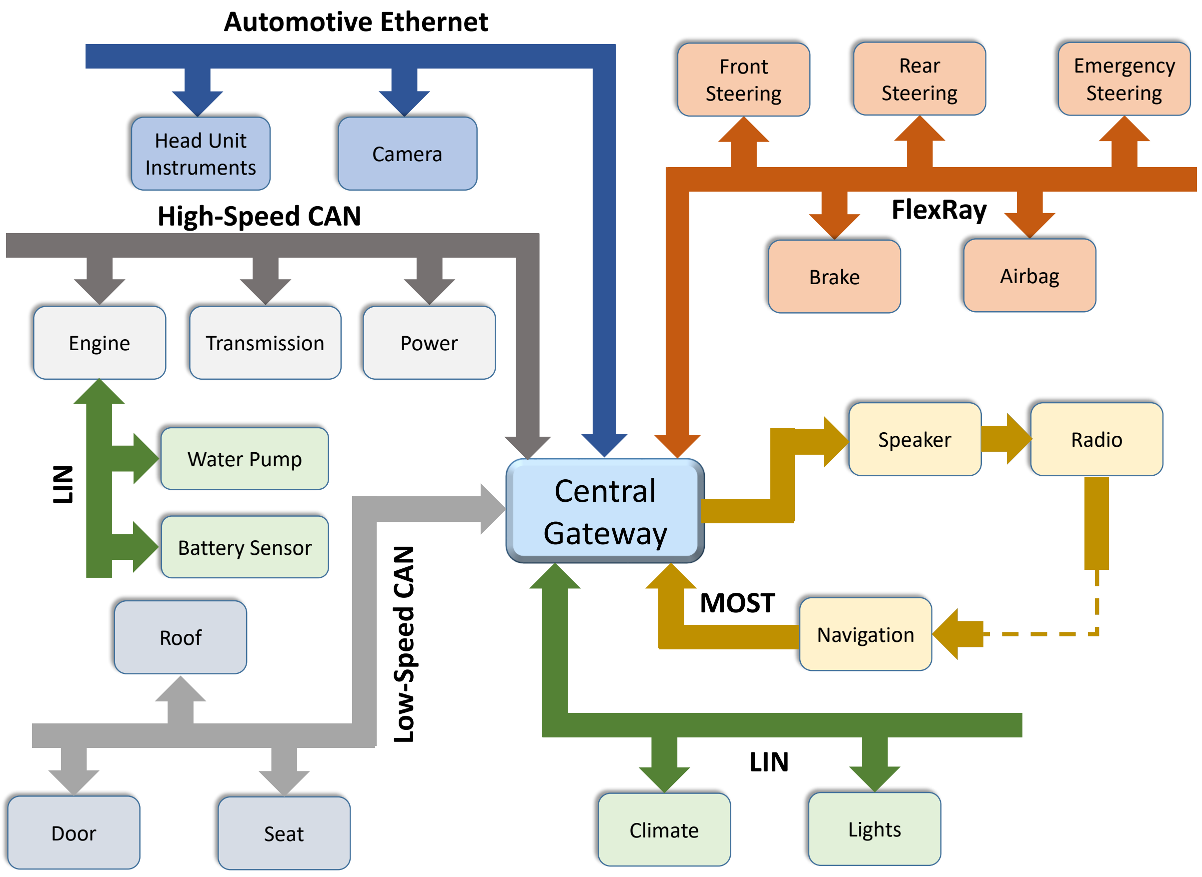


Figure 1.2: An automotive E/E architecture comprising different communication buses including High-Speed and Low-Speed CAN, FlexRay, LIN, MOST, and Automotive Ethernet.

of software tasks only, which is also extended later to OSEK time [17] to support the time-triggered scheduling scheme.

1.1.2 Communication Buses

Automotive E/E systems are composed of several functional domains [9]: (i) Powertrain domain including engine and transmission control; (ii) Chassis domain, e.g., steering and brake control; (iii) Body and comfort domain including the control of doors, seats, lights, HVAC, among others; (iv) Infotainment domain including telematics and entertainment. The large spectrum of software applications has led to diverse requirements from the underlying E/E architecture, in particular, the communication network architecture. The design of automotive in-vehicle networks depends on the nature of data to be transmitted. For example, safety-critical control data in the chassis domain must satisfy real-time properties and reliability requirements while infotainment and camera-based driver assistance data require high network bandwidth [18]. As a result, the E/E system has several subnetworks connected via gateways, as shown in Figure 1.2. These subnetworks offer different network bandwidth and implements different communication protocol. Each subnetwork serves several applications belonging to a specific functional domain, e.g., FlexRay or High-Speed CAN for the chassis domain, High-speed CAN for the powertrain domain, Low-Speed CAN and LIN for the body domain, and MOST and Ethernet for the infotainment domain [19].

FlexRay: FlexRay communication bus is a deterministic, fault-tolerant and high-speed bus system developed by the FlexRay consortium. FlexRay protocol v2.1 [20] was first published in 2005 followed by FlexRay protocol v3.0 [21] in 2010. FlexRay is a hybrid communication protocol, in which each communication cycle is partitioned into a static and a dynamic segment. The static segment is composed of equal length time-division multiple access (TDMA) slots. Each data frame mapped onto a static slot will be sent exactly within the assigned slot. Due to time-deterministic communication in the static segment, it is used for the transmission of safety-critical data with strict timing requirement [13]. On the other hand, dynamic segment is composed of equal-length minislots where the size of a slot is larger (10 times or more) than that of a minislot, and a frame mapped to the dynamic segment can take more than one minislot for transmission. Thus, the dynamic segment offers flexible TDMA (FTDMA) where each frame is assigned a slot number, however, the slot counter is incremented only after a message is completely sent or after an empty slot. Thus, the exact time when a frame is transmitted in the dynamic segment depends on the size of the frames that are assigned lower slot numbers. While the static segment allows time-deterministic communication, the communication in the dynamic segment can have jitters (i.e., there is a best- and a worst-case delay). On the other hand, if there is no new data to be sent on a static slot the whole slot is wasted, while for the dynamic segment, in the absence of a new data, only a minislot is wasted. Thus, the dynamic segment is more resource-efficient than the static segment. As FlexRay supports TDMA, the clocks of all FlexRay nodes in a network is synchronized, and therefore have the same notion of time. FlexRay bus offers two communication channels that can be either used to multiply bandwidth or for redundancy [22]. In each channel, a data rate of 10 Mbit/s is offered. Despite the advantages, the usage of FlexRay in automotive systems is mostly limited to the chassis domain due to the higher cost and complex parametrization when compared to CAN [13].

Controller Area Network (CAN): CAN [23] has been the de facto standard for in-vehicular communication since it was introduced by Robert Bosch GmbH in 1986. CAN offers the maximum data rate of 1 Mbit/s. *High-Speed CAN* [24] with a common data rate of 500 kbit/s and *Low-Speed CAN* [25] with a common data rate of 125 kbit/s are commonly used in automotive systems. CAN frames can traditionally support a payload size of 8 bytes. However, this limits its usage in advanced driver-assistance systems (ADASs), and therefore, CAN Flexible Data-Rate (CAN FD) [26] has been developed that allows a payload of size up to 64 bytes. CAN allows event-triggered communication, where a collision between two CAN frames is resolved using carrier-sense multiple access/collision resolution (CSMA/CR) [27]. That is, each CAN frame is assigned a priority apriori and when two CAN data frames are sent simultaneously, the higher-priority frame will transmit first while the lower-priority one waits. There have been efforts to extend CAN to allow time-deterministic communication in time-triggered CAN (TTCAN) [28]. However, due to limitation of the data rate, TTCAN is not widely used in modern automotive systems [29].

Local Interconnect Network: LIN [30] is a time-triggered low-speed bus that has been developed as a cheaper alternative to CAN bus to connect to sensors and actuators in the body domain for the control of doors and seats [13]. LIN offers a maximum data rate of 20 kbit/s and is based on master/slave communication. In a LIN bus cluster, the master node polls each slave node periodically using an empty frame, where the header contains the id (or number) of the slave being polled and the slave adds data to the empty frame [31]. There is no collision possible in this polling-based communication because each slave has a different id, and therefore, only one slave responds at a time.

Media-Oriented Systems Transport (MOST): It is mainly used for multimedia and telematics applications. It supports high data rates of 25 Mbit/s (MOST25), 50 Mbit/s (MOST50), and 150 Mbit/s (MOST150) [32]. It supports both time-triggered (i.e., audio and video data) and event-triggered (i.e., navigation data) communication [13, 31]. It also follows a master-slave communication like LIN, where a dedicated master ECU creates the message frames in which the slave ECU can send their messages.

Automotive Ethernet: Ethernet has been widely used in local area networks (LANs), metropolitan area networks (MANs), and wide area networks (WANs). However, the traditional Ethernet protocol cannot be deployed in automotive systems due to the following reasons: (i) it is susceptible to noise from other electronic devices, (ii) it cannot provide timing guarantees in the order of microseconds, (iii) it supports only asynchronous communication, and (iv) there is no means to prioritize communication traffics.

The first shortcoming, i.e., higher communication noise in high-speed Ethernet, is addressed by using *BroadR-Reach* [33], the standard for the physical layer developed by Broadcom for Ethernet. *BroadR-Reach* uses a robust and efficient three-level pulse-amplitude modulation (*PAM-3*) signaling scheme that has a high spectral efficiency and therefore, enables a lower signal bandwidth (33.3 MHz). This reduces crosstalk and and also ensures that electromagnetic interference (EMI) requirements of automotive industry is met [34].

The primitive Ethernet protocol is based on carrier-sense multiple access with collision detection (CSMA/CD) with half-duplex links, which is not time-deterministic. In CSMA/CD, when multiple Ethernet frames are sent at the same time, they will collide, and then each Frame is sent again after a random back-off period [35]. Thus, there is no guarantee on how many maximum retransmissions are needed to send a frame reliably.

Towards a more reliable communication, first, full-duplex links are used to allow simultaneous bi-directional communications [36] and, second, switches are introduced in the communication path to forward frames. In full-duplex switched Ethernet network [37], only two nodes share a communication link, and therefore, there is no collision between Ethernet frames. Furthermore, IEEE 802.3br [38], an amendment to Ethernet protocol, make provision for *Express Traffic* or high-priority traffic. High-priority frames can preempt other frames, and thus, have latency of less than a microsecond.

The IEEE 802.1 Audio Video Bridging (AVB) Task Group, later renamed to Time-Sensitive Networking (TSN) Task Group [39], has proposed several amendments to the Ethernet protocol towards addressing further limitations. For example, (i) IEEE 802.1AS provides a standard for time synchronization in Ethernet networks [40], (ii) IEEE 802.1Qat specifies a standard to reserve bandwidth for traffic streams [41], and (iii) IEEE802.1Qav defines a standard for queueing and forwarding of time-sensitive frames [42]. These aforementioned amendments enable time-triggered communication, while the traditional best-effort communication is also supported.

Currently, Ethernet is deployed primarily in the infotainment systems of modern cars and might even replace MOST [13, 43]. However, it is envisioned that Ethernet can serve as the backbone network connecting different application domains [44] and time-triggered communication in Ethernet [45] can also be used for safety-critical systems.

1.1.3 Automotive Software

In recent years, increasingly more software applications are being deployed in the automotive systems, including, e.g., applications for driver assistance systems, infotainment, and safety-critical control systems. This trend is easily observed in the increasing number of ECUs, communication buses, communication signals, lines of software code, among others [46]. However, this trend is not sustainable using the existing federated architecture of the automotive E/E systems, where each ECU runs software pertaining to one application only [47]. Thus, automotive industry is moving towards *ECU consolidation*, where multiple functions can be integrated into one ECU, to efficiently utilize the hardware resources in future automotive systems [48]. However, to handle such software complexity, a sophisticated software architecture needs to be considered [49,50].

AUTomotive Open Software ARchitecture (AUTOSAR): A worldwide partnership of vehicle manufacturers, suppliers, service providers, and companies from the automotive electronics, semiconductor and software industry was formed to develop AUTOSAR [53]. It defines an open industry standard for the automotive software architecture [54]. It specifies the software architectural components, their interfaces, and a standard design and implementation methodology. It allows development of high level application software independent of underlying implementation details. The AUTOSAR ECU software consists of three components, as shown in Figure 1.3. (i) The basic software (BSW) layer provides the basic services (including memory, communication, input-output, system calls, among others) and abstracts the hardware resources by means of drivers and operating system (OS). Here, AUTOSAR-OS is backward compatible with the OSEK-OS. BSW is further partitioned into micro-controller abstraction, ECU abstraction and complex drivers, and services. (ii) The application software (ASW) layer represents the software components that define the functionality of an ECU by means of runnables, e.g., steering control software. (iii) The runtime environment (RTE) layer defines the communication between ASW and BSW. Here, RTE allows development of application software independent of hardware as it contains the details about when the runnables use the drivers. Therefore, if a runnable is removed or added then the RTE must also change. This implies that the basic software and RTE along with all existing applications have to be rebuilt in case a new application is added in AUTOSAR classic platform [51].

Besides, the software components, AUTOSAR also defines the communication interfaces between the components as *ports*. There are different types of ports (e.g., sender-receiver and client-server) realizing different communication relations. The communication between components is realized based on the configuration of the ports via the *virtual functional bus* (VFB) [52]. VFB abstracts the actual implementation of the communication that depends on the mapping of the software components, i.e., VFB can represent communication bus systems if the soft-

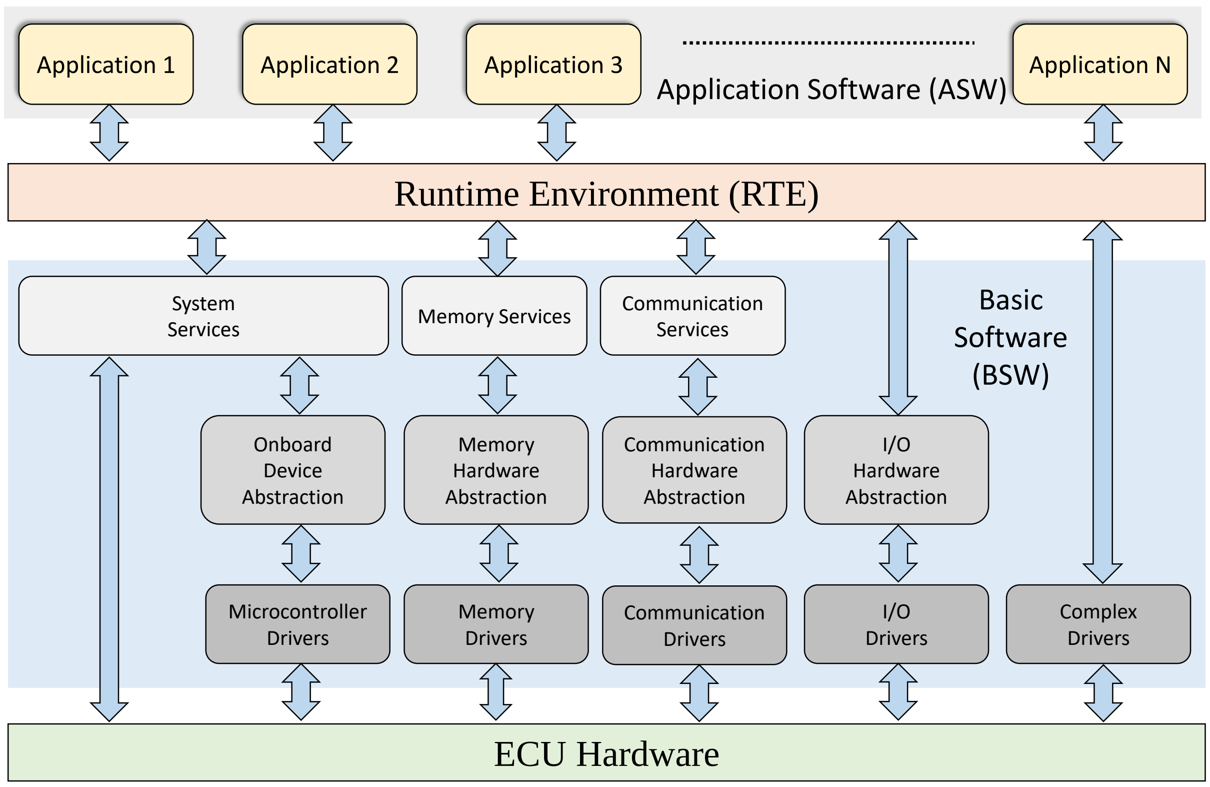


Figure 1.3: The automotive software architecture as defined by AUTOSAR. It consists of three layers, namely, the application software (ASW), the runtime environment (RTE), and the basic software (BSW). This figure is reproduced from [51] and [52].

ware components are mapped on different ECUs, while it can even represent communication via RTE and the local memory if the software components are in the same ECU. Complying with AUTOSAR standard makes automotive software highly modular, thus reducing the complexity while also allowing reusability. Thus, the basic software and the application software can be developed by different suppliers and then integrated by the original equipment manufacturer (OEM). Moreover, different applications mapped on to the same ECU can be developed by different suppliers. Even an application software can be reused for a new vehicle variant with different E/E architecture by properly configuring the RTE.

1.1.4 Battery Packs

Towards a pollution free and sustainable transportation solution, battery packs are used in EVs and hybrid electric vehicles (HEVs) as a source of energy [55]. Batteries are widely used to store electrical energy due to their high energy (and power) density and high specific energy (and specific power), which correlates to a lower required installation volume and a lower weight respectively, compared to other electrical energy storage (EES) solutions such as supercapacitors [56]. Moreover, batteries can be tailored to meet the specific requirements of the application such as fast charging, longer shelf life, and higher power rating [55].

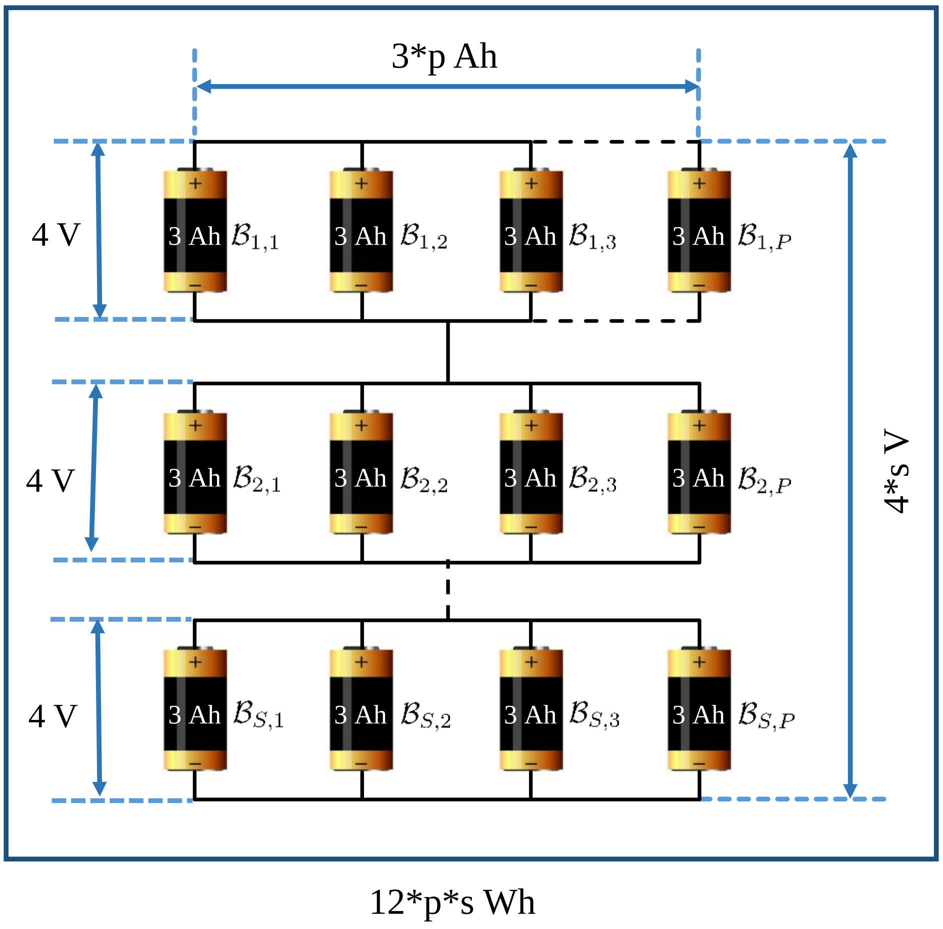


Figure 1.4: Series and parallel connection of electrochemical cells to form high-power battery packs. Series connection enables the pack to operate at high voltage while parallel connection increases the capacity to drive higher load currents. Here, p cells are first connected in parallel to form a group and then s such groups are connected in series.

Batteries are composed of electrochemical cells [57], where a chemical reaction can cause an electron transfer. These cells consist of a positive electrode (i.e., cathode), a negative electrode (i.e., anode), and an electrolyte that facilitates the movement of charge carriers between electrodes. In general, batteries are broadly classified into primary (non-rechargeable) and secondary (rechargeable). The primary non-rechargeable batteries comprise galvanic cells that can be used only once. They are discarded when the active chemical materials (generating electricity) in the cells are fully utilized. By contrast, the secondary rechargeable batteries comprise cells that can operate both as galvanic and electrolytic cell and are therefore, based on reversible chemical reactions. The secondary batteries can be charged and discharged several times. During charging, positive ions ($M+$) are generated at the cathode that move to the anode through the electrolyte. During discharging, the opposite reaction takes place where positive ions move from the anode to the cathode. Note that electrons (e^-) move in the opposite direction to that of the ions through the outer circuit, which results in the charging and the discharging current in the respective phases. For EES, the secondary rechargeable batteries are preferred since they allow to store (charge) and use (discharge) the electrical energy without the necessity to replace the battery itself.

Several different cell chemistries are available for rechargeable batteries [58, 59]. Examples include: (i) Nickel-metal hydride (NiMH) cells consist of nickel oxide hydroxide as the cathode, a hydrogen absorbing alloy as the anode, and potassium hydroxide as the electrolyte. (ii) Lead-acid cells have lead-dioxide as the cathode, metallic lead as the anode, and sulphuric acid solution as the electrolyte. (iii) In lithium ion (Li-ion) cells, the cathode is a lithium compound (e.g., lithium magnesium oxide, lithium cobalt oxide, and lithium iron phosphate), the anode is typically graphite, and the electrolyte comprises lithium salt in an organic solvent.

Compared to other rechargeable cell chemistries, batteries that consist of Li-ion cells have the following advantages [60]: (i) They perform better in terms of energy and power densities because the electrochemical potential of lithium is higher compared to other materials. Therefore, Li-ion cells can be manufactured in smaller size and weight for the same energy and power requirements of the application. Additionally, Li-ion cells can be manufactured in several shapes, and therefore, can be organized in a battery pack compactly. (ii) Li-ion cells have no memory effect and thus have a longer cycle life compared to NiMH and nickel-cadmium (NiCd) where, due to the memory effect, energy capacity reduces with repeated charging after partial discharges. (iii) Li-ion cells are generally stable to high currents. Thus, battery packs composed of Li-ion cells are typically used in EVs and HEVs, where high currents are experienced during acceleration and braking. (iv) Li-ion cells have a high coulombic efficiency and a low self-discharge rate, and thus, they are energy-efficient.

Typically, battery packs for EVs and HEVs must operate at a high voltage ($\approx 450\,\mathrm{V}$) and have a high capacity ($\approx 200\,\mathrm{Ah}$). However, the voltage and capacity of a single Li-ion cell is insufficient to meet the aforementioned requirements. Therefore, automotive battery packs are composed of a number of series- and parallel-connected Li-ion cells, as shown in Figure 1.4. In order to drive a higher current, multiple Li-ion cells are connected in parallel and the high operating voltage is obtained by series-connection of the cells [11]. Examples of automotive battery packs include: (i) The Tesla Model-S uses a $85\,\mathrm{kWh}$ battery pack [61] that comprises Panasonic “18650” Li-ion cells where each cell has a capacity of $3.2\,\mathrm{Ah}$. The battery pack is composed of 16 series-connected module and each module consists of 6 series-connected groups of 74 cells that are connected in parallel. (ii) The battery pack in Nissan Leaf has 48 series-connected modules where each module has 2 groups of 2 cells in parallel [62]. Here, sheet-shaped Li-ion cells are used that have a cell capacity of $32.5\,\mathrm{Ah}$. (iii) The BMW i3 battery pack comprises 8 series-connected modules where each module has 12 series-connected cells [63]. Here, $60\,\mathrm{Ah}$ prismatic cells are used.

1.1.5 Battery Management Systems

Typically, battery packs used in high-power automotive applications are provided with a hardware/software system, i.e., the *battery management system* (BMS), to maintain their safety and performance [64]. Despite the advantages offered by the Li-ion cells, they are sensitive to their operating conditions in terms of voltage, current, and temperature, and thus, these parameters must be tightly monitored and controlled to ensure safety [65]. Furthermore, the state of the

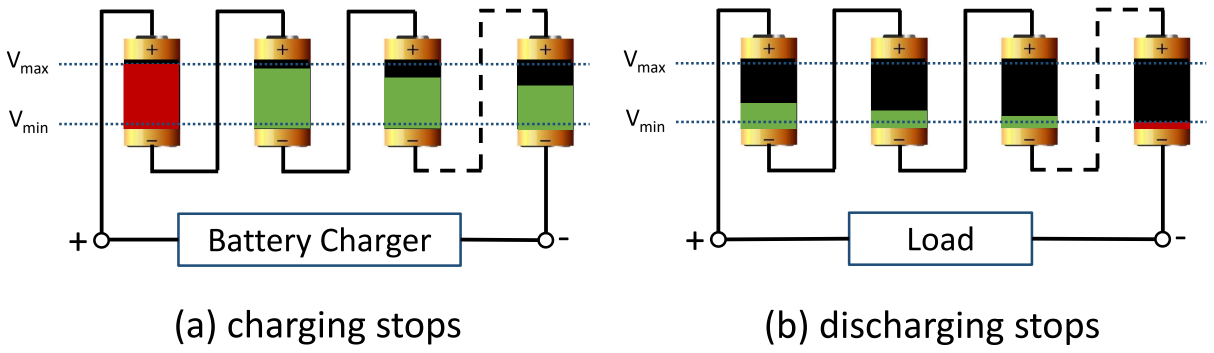


Figure 1.5: Charge variation in a battery pack results in reduced usable capacity. (a) A battery pack with series-connected cells cannot be charged further, if a cell has reached the maximum threshold of charge, despite there are cells in the pack that are not fully charge. (b)A battery pack with series-connected cells cannot be discharged further, if a cell has reached the minimum threshold of charge, despite there are cells in the pack that can still drive current.

cells i.e., the state-of-charge (SoC) and the state-of-health (SoH), in a battery pack must be controlled such that the maximum performance (i.e., energy output) can be obtained [64].

Safety concerns [65]: Li-ion cells have a defined set of safe operating conditions and operation outside the specified limits can severely damage the cells, reduce their lifetime, and even cause fire or explosion due to thermal runaway. (i) The minimum and maximum operating voltage of most Li-ion cells are in the range of 2.7 V and 4.2 V respectively. Charging a Li-ion cell with a voltage higher than that specified can cause *lithium plating* on the anode surface, reducing the availability of lithium ions and thus, resulting in an irreversible capacity loss. Lithium plating can also lead to an internal short-circuit of the cell causing excessive temperature that damage the cell. Similarly, discharging a Li-ion cell below its minimum threshold voltage results in a gradual breakdown of the internal cell electrodes, reducing their lifetime. (ii) The maximum current with which a cell can be charged depends upon the design of the electrodes. Increasing the charging current significantly increases the temperature of the cell. In certain cases, the cells experience an increased pressure at higher currents and start to swell due to generation of gases inside the cell. If the mechanical clearance between cells in the pack is not adequate, it will result in a short circuit situation. Likewise, discharging the cell with higher currents results in an inherent capacity loss, which is referred to as the *rate capacity effect*. (iii) Temperature influences the safety and performance of the battery. On one hand, at low temperatures, the speed of chemical reactions slows down that results in a reduced current carrying capacity of the pack. Prolonged operation at reduced temperatures (below 0 °C) will result in a premature capacity loss of the pack. On the other hand, increased temperatures will result in catastrophic effects causing fire or explosion.

Performance concerns: As mentioned in Section 1.1.4, automotive battery packs comprise multiple series-connected Li-ion cells. The usable capacity (or the maximum energy output) of such a pack is limited by the cell with the minimum charge level or SoC, compared to other series-connected cells [66]. Thus, it is desirable to charge each cell to the maximum SoC threshold to maximize the usable capacity. However, the charging process stops when a cell in the pack reaches the maximum SoC threshold despite other cells not being fully charged [67], as shown in Figure 1.5a. Similarly, during discharging, when a cell reaches the minimum SoC threshold, no more power can be drawn, despite there is residual charge in the other cells of the pack, as illustrated in Figure 1.5b. Note that when all cells in the battery pack have identical charge/discharge characteristics, this imbalance in the SoCs is not observed. However, realistically, there is a variation in the SoCs primarily due to the following reasons: (i) There might be a variation in cell characteristics (up to 10%) due to manufacturing variances [68, 69]. A Li-ion cell is manufactured by mixing granular raw materials to form a slurry [70]. The variation in the shape or size of the grains or in the composition of the slurry can lead to different characteristics of the manufactured cells [71]. These differences become more profound when these cells age. (ii) Non-uniform temperature distribution is also a major reason for the variation in the SoC levels in the pack. It has been observed that the temperature closer to the cooling inlet is more than 10 °C lower than the temperature farthest away [72]. Arrhenius' law states that the rate of chemical reactions doubles with every 10 °C rise in temperature [73]. Thus, with non-uniform temperature distribution, each cell will have a different discharge characteristic. Moreover, the internal resistance and the self-discharge rate of a cell also vary with temperature [74].

BMS tasks: Towards the aforementioned safety and performance concerns in a battery pack, the BMS is required. Typically, the BMS performs the following tasks [75]: (i) A BMS comprises sensors to read cell parameters, i.e., voltage, current and temperature. Based on the sensor values, it maintains the operation of the battery pack within the safety limits. For example, it co-ordinates with the HVAC control unit to regulate the temperature. It also prevents overcharging by communicating with the charging station. (ii) Typically, BMS performs cell balancing to equalize the charge levels of the series-connected cells in the battery pack. In passive cell balancing, it determines the amount of charge that each cell must dissipate across the resistor. In active cell balancing, the excess charge needs to be redistributed, and therefore, it runs the control algorithm that determines the charge transfer pairs. In addition, it also generates the pulse-width modulation (PWM) signals to drive the power-electronic switches that enable charge transfer from the source to the destination cells. (iii) BMS computes the SoC of each cell in the pack. SoC values are required to perform cell balancing, which maximizes the usable capacity of the battery pack and therefore increases the driving range of the EV. BMS also needs to estimate the SoH of the battery pack, i.e., the ability to deliver energy. It is extremely important to accurately estimate the SoH such that the battery can be replaced at an appropriate time without inconveniencing the car user.

1.2 Motivation and Challenges

Designing a CPS can generally be described as a multidisciplinary engineering process [76–78]. A general design flow of a CPS can be described as follows:

- The design starts with the mathematical model of the physical system provided by the process engineers (e.g., mechanical, electrical, production, chemical, and/or biomedical engineers). For example, if a medical device is being developed, the biomedical engineer would have the necessary knowledge of the physical process, while for a complex cyber-physical manufacturing system, mechanical, electrical, and production engineers come together to develop the process model.

- Based on the process model, the control engineer develop the control algorithm to meet certain stability and performance requirements. The control engineer might not have sufficient knowledge of the implementation platform. In such a case, certain assumptions are made to design the controller. Later, if these assumptions are not realized by the implementation, controller redesign might be necessary.

- The codes generated from the control algorithm are then integrated with other software components (including operating systems, driver software, interfacing with peripheral devices, interrupts, among others) by the software developer. When multiple applications share platform resources, this software integration is supervised by the systems engineer. Systems engineers have a good overview of the system and decide on the system architecture, i.e., the mapping of the software components on the ECUs, the physical layout of the ECUs and their interconnections.

- Each ECU is then analyzed by specialized computer engineers for timing and functional correctness.

- A group of network engineers designs the communication systems, i.e., mapping data frames on to different communication links, scheduling the frames and analyzing communication timings. Each communication system is designed by a specific group trained on the corresponding communication protocol.

- At different stages of the design process, test engineers perform model-in-the-loop (MIL), software-in-the-loop (SIL), and hardware-in-the-loop (HIL) tests to establish functional and timing correctness up to the required standard.

Therefore, CPSs design requires close collaboration between a multidisciplinary team of engineers [79]. However, in the current state-of-practice, collaborations have primarily been ad-hoc and manual [80]. For the design of CPSs, there is a lack of well-established approaches that integrate the models and design philosophies of different engineering disciplines. Therefore, traditional CPSs design approaches have largely been sub-optimal and significant improvements are possible through more comprehensive modeling and design.

1.2.1 Motivation towards Multi-Domain Coupling in CPSs Design

This thesis advocates systematic coupling of multiple interacting domains towards designing safe, high-performance, and cost-efficient CPSs. In this context, we will provide two motivational examples concerning distributed control systems and active cell balancing respectively.

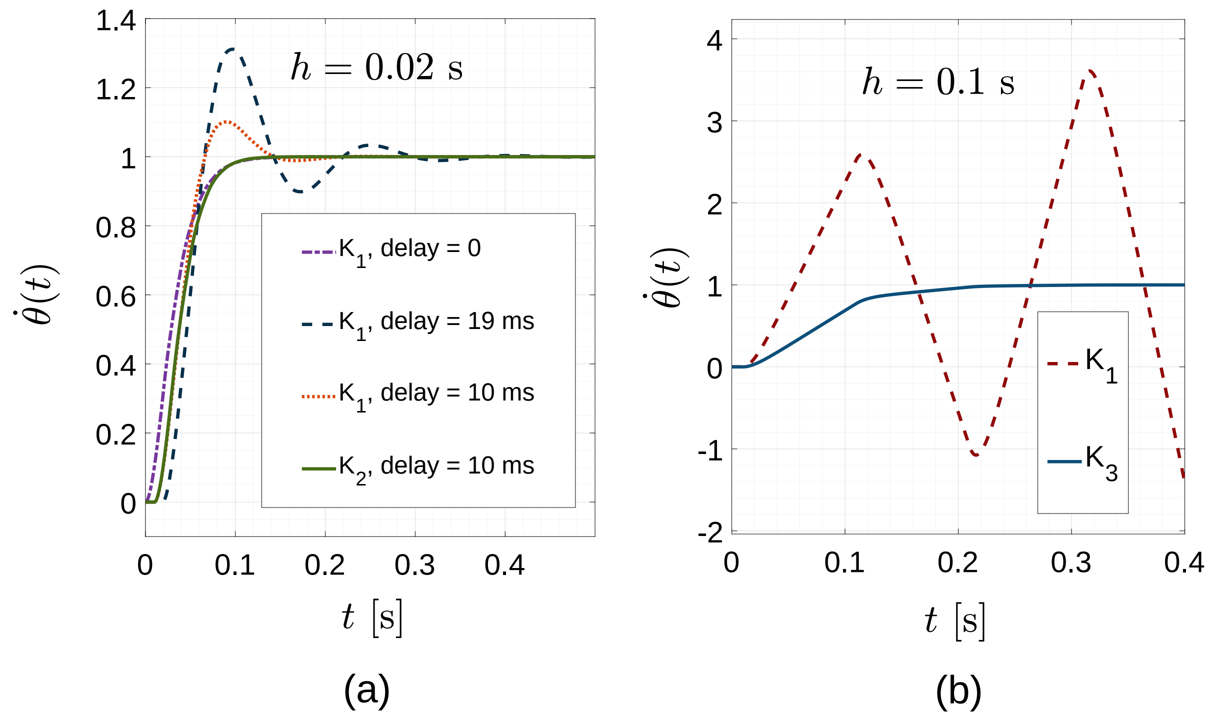


Figure 1.6: Control response depends both on the control law and the software implementation. Three controllers are designed as follows: (i) K_1 is designed for a sampling period of $20\,\mathrm{ms}$ and zero delay; (ii) K_2 is designed assuming a sampling period of $20\,\mathrm{ms}$ and a delay of $10\,\mathrm{ms}$; and (iii) K_3 is designed for a sampling period of $100\,\mathrm{ms}$ and a delay of $10\,\mathrm{ms}$. (a) This figure shows the variation of system response with delay for the same control law K_1. It further shows that a controller K_2 designed appropriately considering the delay of $10\,\mathrm{ms}$ achieve the same performance as obtained with K_1 and zero delay. (b) This figure shows that when K_1 is implemented using a sampling period of $100\,\mathrm{ms}$, it can result in an unstable system. However, the controller K_3 designed appropriately for $100\,\mathrm{ms}$ can stabilize the system.

1.2.1.1 Distributed control systems: A motivational example

We study a DC motor speed control system [81]. The physical dynamics of the motor in continuous time is given by the following equation:

$$\begin{bmatrix} \ddot{\theta}(t) \\ \dot{i}(t) \end{bmatrix} = \begin{bmatrix} -0.2 & 0.667 \\ -10 & -100 \end{bmatrix} \begin{bmatrix} \dot{\theta}(t) \\ i(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 100 \end{bmatrix} V(t). \tag{1.1}$$

Here, $\theta(t)$ is the angular position of the motor, $i(t)$ is the motor current and $V(t)$ is the motor terminal voltage at a time t. The speed of the motor ($\dot{\theta}$) can be controlled to achieve a reference speed $\dot{\theta}_r$ by varying the voltage applied at the motor's terminal based on the values of speed and current of the motor.

We consider a distributed embedded implementation of the controller. We first assume that the controller samples the system every $20\,\mathrm{ms}$. With the sampling period $h = 20\,\mathrm{ms}$ and in the ideal case of zero delay ($d = 0\,\mathrm{ms}$) between sensing and actuation, the discrete-time dynamics of the system evolve as follows:

$$\begin{bmatrix} \ddot{\theta}[k+1] \\ \dot{i}[k+1] \end{bmatrix} = \begin{bmatrix} 0.9953 & 0.0057 \\ -0.0862 & 0.1349 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} + \begin{bmatrix} 0.0076 \\ 0.8643 \end{bmatrix} V[k], \tag{1.2}$$

where k denotes a sampling instant. We design a state-feedback controller K_1 based on the dynamical model in Eq. (1.2) using standard techniques, where the control law is as follows:

$$V[k] = -\begin{bmatrix} 42.1013 & 0.2452 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} + 42.5747\, \dot{\theta}_r. \tag{1.3}$$

That is, the voltage applied at the motor's terminal can be calculated using Eq. (1.3). We simulate the closed-loop system comprising the motor and the controller assuming zero delay between sensing and actuation to obtain the unit step response, as shown in Figure 1.6(a).

Now, let us assume that the control algorithm is implemented using three tasks T_s, T_c and T_a, where the execution times of the tasks are $1ms$, $3ms$ and $1ms$ respectively. Here, (i) T_s reads the sensor values, i.e., speed and current, (ii) T_c computes the control input $V[k]$ using the control law, and (iii) T_a applies the input to the motor using a voltage regulator. Thus, the start times of the task T_s co-incide with the sampling instants while the actuation takes place at the end of the task T_a. The tasks are mapped on different ECUs, each with a time-triggered scheduler. The data between T_s and T_c is transmitted as a message m_s and the data between T_c and T_a is transmitted as another message m_c. The messages are transmitted over CAN, which exhibits fixed-priority non-preemptive scheduling. Now, let us consider two scenarios where the worst-case delays in m_s and m_c are respectively given as (i) Case 1: $2ms$ and $3ms$, and (ii) Case 2: $6ms$ and $8ms$. For both cases, we schedule the tasks such that the events from sensing-to-actuation (T_s, m_s, T_c, m_c, and T_a) are in the correct temporal order. In the first implementation, the sensing-to-actuation delay d is $10\,\mathrm{ms}$, while it is $19\,\mathrm{ms}$ in the second implementation. This clearly violates the assumption of zero delay that we have made while designing the control law given in Eq. (1.3).

Now, we simulate the closed-loop system for a unit step reference signal with sensing-to-actuation delay of $10\,\mathrm{ms}$ and $19\,\mathrm{ms}$ respectively. The response curves for these simulations are depicted in Figure 1.6(a). For the delayed systems, note that the control input $V[k]$ is computed using Eq. (1.3), based on the values of speed and current obtained in the k-th sampling instant. However, $V[k]$ is applied d time units after the k-th sampling instant. It may be noted that the overshoot and the settling time have increased from the ideal case with no delay. Thus, we may say that the performance has deteriorated, and more importantly, it does not match the expected values that were obtained during the controller design stage. This performance degradation can be attributed to the delays introduced in the loop which we will study in more detail in Section 2.1 in Chapter 2. Similarly, when the controller K_1 is implemented using a sampling period of $100\,\mathrm{ms}$ and a sensing-to-actuation delay of $10\,\mathrm{ms}$, it cannot stabilize the system as shown in Figure 1.6(b). Thus, a controller is not reliably designed if the timing properties of the platform implementation are not accounted in.

Safety properties of embedded control systems is usually captured in terms of stability and performance guarantees. We can observe in the above example that when the controller is designed separately from the embedded platform and is oblivious to the implementation details, no such guarantees at the implementation level are possible. In practice, costly testing and integration efforts are necessary to obtain an acceptable implementation by iteratively changing the controller model and the platform parameters.

There is a mismatch between the actual response and the expected response. This is because the dynamical model of the closed-loop system changes with differnt platform implementations. For example, for a sampling period $h = 20\,\mathrm{ms}$ and a delay $d = 10\,\mathrm{ms}$, the discrete-time system model becomes:

$$\begin{bmatrix} \ddot{\theta}[k+1] \\ \dot{i}[k+1] \end{bmatrix} = \begin{bmatrix} 0.9953 & 0.0057 \\ -0.0862 & 0.1349 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} + \begin{bmatrix} 0.0025 \\ 0.6321 \end{bmatrix} V[k] + \begin{bmatrix} 0.0051 \\ 0.2323 \end{bmatrix} V[k-1]. \tag{1.4}$$

Likewise, in case of $h = 100\,\mathrm{ms}$ and $d = 10\,\mathrm{ms}$, the dynamical model is given as follows:

$$\begin{bmatrix} \ddot{\theta}[k+1] \\ \dot{i}[k+1] \end{bmatrix} = \begin{bmatrix} 0.9743 & 0.0065 \\ -0.0977 & -0.0006 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} + \begin{bmatrix} 0.0528 \\ 0.9952 \end{bmatrix} V[k] + \begin{bmatrix} 0.0065 \\ -0.0006 \end{bmatrix} V[k-1]. \tag{1.5}$$

Both Eq. (1.4) and Eq. (1.5) are different from the model in Eq. (1.2).

Now, we design controllers K_2 and K_3 based on the dynamical models in Eq. (1.4) and Eq. (1.5) respectively. K_2 can be written as follows:

$$V[k] = -\begin{bmatrix} 43.9035 & 0.2946 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} - 0.2363\, V[k-1] + 44.4862\, \dot{\theta}_r. \tag{1.6}$$

K_3 is given by:

$$V[k] = -\begin{bmatrix} 12.666 & 1.1146 \end{bmatrix} \begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix} + 0.9546\, V[k-1] + 13.0184\, \dot{\theta}_r. \tag{1.7}$$

We simulate the closed-loop system with K_2 and K_3 according to their respective design assumptions and the system responses are given in Figure 1.6(a) and Figure 1.6(b) respectively. It can be observed that using K_2, we can obtain a better response than K_1 for the same implementation. This shows that control performance can be improved if platform details are appropriately considered during the controller design. On the other hand, Figure 1.6(b) shows that the control response using K_3 is stable. Moreover, if the performance of K_3 is acceptable according to the design specification, the implementation of K_3 with a higher sampling period of $100\,\mathrm{ms}$ is highly efficient (5 times) in terms of resource usage. This is because the tasks are executed and the messages are sent only once in $100\,\mathrm{ms}$. With a sampling period of $20\,\mathrm{ms}$, the same tasks and messages are repeated 5 times within a period of $100\,\mathrm{ms}$. For the cases where the design assumptions are preserved in the implementation, note that K_3 does not perform as good as K_1 and K_2. This shows that the reduction in resource usage comes at the cost of performance and there exists a trade-off between the two.

Summary: This example shows that by studying the integrated models of the physical system, the control law, and the platform implementatiion of the control software, it is possible to design a controller that is stable, offers good control performance, and uses less resources.

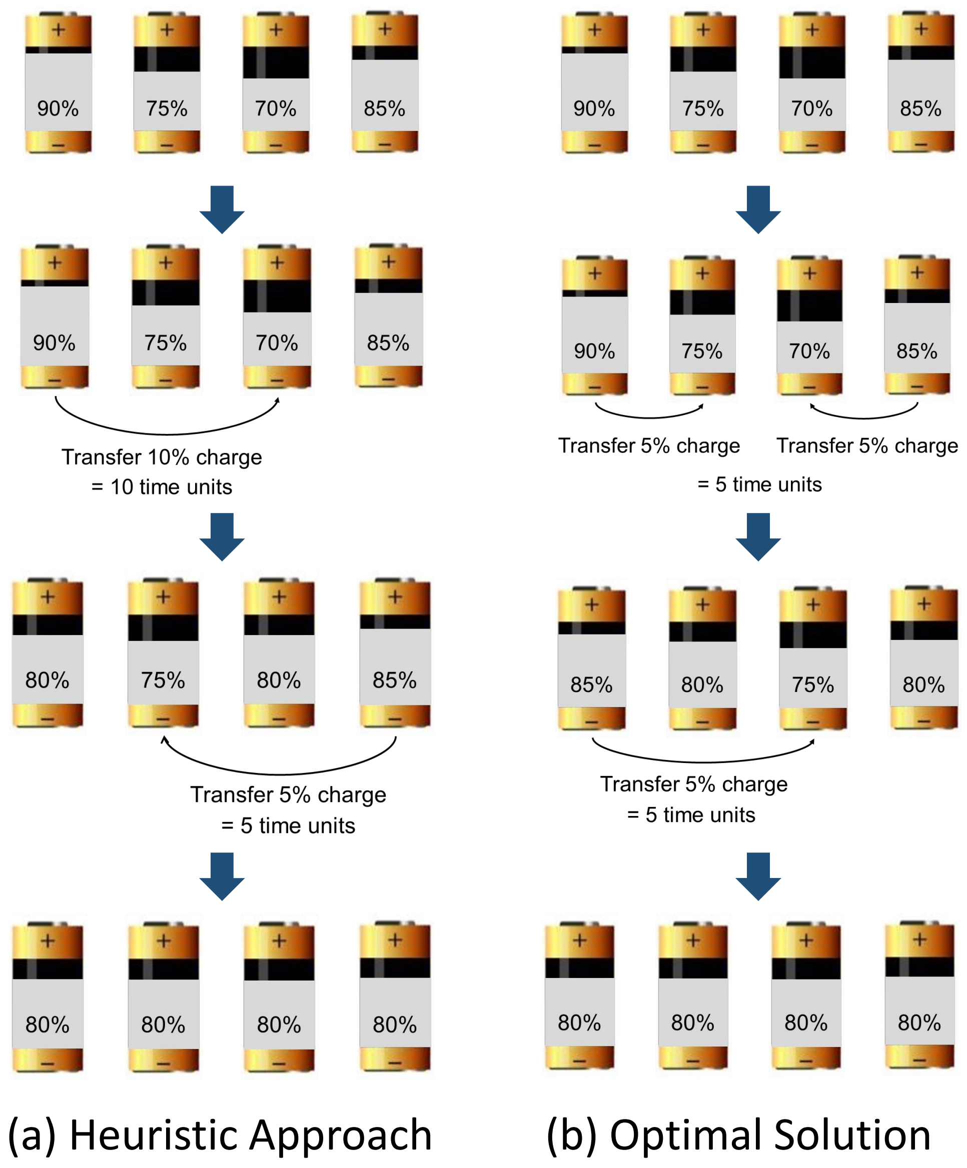


Figure 1.7: Optimal cell balancing vs existing heuristics. The heuristic takes 15 time units for cell balancing while the optimal approach takes only 10 time units. The heuristic selects non-neighbor charge transfer pairs of cells more often than the optimal approach, however, transfering charge over a larger distance results in more energy dissipation.[2]

1.2.1.2 Active cell balancing: A motivational example

In active cell balancing, charges are transferred between cells in the pack until the SoCs of the cells are within a certain threshold (i.e., approximately equalized). Thus, a cell balancing algorithm selects charge transfers and schedule them such that the SoCs of the cells are equalized eventually. However, there are numerous combinations of charge transfers that can lead to a

[2]The values presented in this figure are only approximate for the ease of understanding and not based on profound mathematical models of charge transfer.

balanced battery pack. For selecting a combination, the two main objectives are to minimize energy dissipation and balancing time respectively [82–84].

Let us take an unbalanced battery pack of 4 cells with charge levels at 90 %, 75 %, 70 %, and 85 % respectively, in order of their serial connection. Here, charge is measured as a percentage of full charge that a cell can hold. Now, consider a heuristic that selects the weakest cell among the set of available cells as the charge receiver. Then, in the neighborhood of the receiver cell, it looks for the donor cell that has the maximum charge level among the cells with which it can be feasibly paired. For charge transfers, we consider the circuit architecture explained in [85]. In this architecture, charge is transferred between cells via a charge transfer bus where concurrent charge transfers are allowed, however, with certain restrictions. The details of this architecture are provided in Section 2.3.

Using the heuristic, two charge transfers are successively selected in the aforementioned example for cell balancing as shown in Figure 1.7(a). To begin with, the third cell has the minimum charge at 70 % and therefore is selected as the receiver cell, while the first cell is the donor with maximum charge at 90 %. The first cell transfers 10 % charge to the third cell until their charge levels become equal at 80 %. During this charge transfer, the second cell becomes unavailable for simultaneous charge transfer because the part of the bus from the first cell to the third cell is already in use and no two charge transfers can have overlapping current flow paths to prevent a short-circuit condition. This constraint is discussed in detail in Section 6.3.2.1. Although the second and fourth cell are not equalized, they have to wait until the charge transfer from the first to the third cell is concluded. After having waited, the second cell becomes the receiver with minimum charge level at 75 % while the fourth cell is the donor with maximum charge level at 85 %. Thus, 5 % charge is transferred from the fourth cell to the second cell till they become equalized at 80 %. Now, all four cells in the pack have 80 % charge and the pack is balanced. Assuming that 1 time unit is required to transfer 1 % charge, the total time taken by the heuristic is 15 time units. Moreover, note that 15 % charge is transferred between non-neighbors. Non-neighbor charge transfers have more energy dissipation compared to neighbor transfers due to more resistances in the current flow path accounting for more switches and longer wires. More details on energy dissipation will be provided in Section 2.3 and Section 6.3 respectively.

Now, the question is: Is there a charge transfer schedule that is more optimal with respect to the design objectives of energy dissipation and balancing time? Here, we consider an alternative solution as depicted in Figure 1.7(b). This solution first schedules two charge transfers in parallel, i.e., the first cell and the fourth cell transfer 5 % charge to the second cell and the third cell respectively. After these transfers, the first cell transfers 5 % charge to the third cell, thereby, resulting in a balanced battery pack. In this solution, 15 % charge is transferred in 10 time units by scheduling parallel transfers, thus, the solution is time-efficient. On the other hand, 10 % charge is transferred between neighbors that is more energy-efficient. This solution is better than the heuristic both in terms of resource usage (i.e., less energy dissipation) and control performance (i.e., lower balancing time) respectively.

Summary: This example shows that a heuristic solution to cell balancing can be sub-optimal if it is oblivious to the physical models of charge transfer between cells and the constraints of the underlying charge transfer circuit architecture. A more integrated modeling and advanced design space exploration is necessary for an optimal control of the cell balancing process.

1.2.2 Challenges towards Multi-Domain Coupling in CPSs Design

It is established in the last section that multi-domain coupling is necessary to synthesize safe, high-performance and resource-efficient CPSs. However, there are several challenges to be addressed before multi-domain coupling can be realized for industrial-sized systems [86, 87].

- CPSs comprise heterogeneous models [88,89], coming from different engineering disciplines, that might not be trivial to integrate. For example, a physical system evolves continuously according to a set of differential equations while the corresponding controller might be dispatched periodically and after each dispatch it contest for the resource based on a certain scheduling policy that can be represented using a finite state machine. An emerging approach to model such a hybrid system uses finite state automata, where system properties like reachability and safety expressed as linear temporal logic (LTL) formulae can be verified using well-established theories of computer science like model checking and game theory. Although there has been sufficient emphasis on studying “hybrid control systems” [90–92], existing solutions are still not scalable to realistic industrial systems.

- Design philosophies traditionally followed in different design domains are also very different from each other. For example, a controller is designed by finding the correct set of parameters like closed-loop poles, sampling period and delay, which optimize the QoC. Due to non-linear dependence of QoC on these parameters and platform induced constraints on them, the control design problem does not fit into any standard closed-form framework for optimal control. Therefore, one needs to search for the optimal pole values either exhaustively or using heuristics. Moreover, the controller for each application is designed separately. On the other hand, the platform design problem is often formulated as a constraint programming model. It must consider all applications that are mapped on the shared platform. The designed values of the sampling periods and delays from the control design stage can be translated into constraints for the platform design. However, in case a feasible solution does not exist, it is difficult to systematically determine which controllers need to be redesigned. An even more difficult question is what is the optimal way of redesigning the controllers to obtain a feasible platform configuration [93].

- Integrated design of CPSs consists of large number of design dimensions coming from different design domains. For example, a combined design space of controllers and platform parameters is huge and challenging to handle. Such synthesis problems require considerable computational effort. This is aggravated by the increase in the system size. Therefore, the scalability of the design space exploration (DSE) technique is an important criterion to be considered before applying it to industrial-sized systems.

- The requirements from different design domains might be conflicting to each other. For example, control engineers try to design optimal controllers with respect to QoC while systems engineers aim to minimize the overall system cost pertaining to resource usage. In most cases, the more is the resource attributed to a control application in terms of computation time and communication bandwidth, the higher is the QoC. Therefore, it is important to study the trade-off between the two design objectives. It is also desirable to allocate resource to applications in a way such that an optimal trade-off between QoCs of different applications is achieved. Thus, the integrated design of controllers and platform parameters translates into a non-trivial multi-objective optimization problem.

- CPSs design typically involves a disjoint set of tools offered by different suppliers [94]. Each tool is a product of years of experience in specific domain. Tool developers mostly have a particular set of expertise. Thus, it is challenging to extend one tool and incorporate the functionalities of the other (from a different domain) without compromising the quality. Correspondingly, an integrated toolchain will require a strong collaboration among tool developers from different domains.

In this thesis, we explain our efforts to address these aforementioned challenges for a more comprehensive design of distributed control systems and cell balancing algorithms.

1.3 Evolution of Design Approaches for CPSs

Over the last 10 years, the concept of CPS emphasizes the integrated modeling and analysis of computational platforms and the physical processes that are controlled by such platforms. In this section, we briefly study the evolution of design paradigms for CPSs. For a more detailed study, please refer to [5]. Here, we first study the problems with separation of concerns, where controller design and platform implementation are carried out in isolated design spaces without sufficient knowledge of each other. Subsequently, we discuss the CPS-oriented approaches and broadly classify them in terms of whether (i) the implementation platform is fixed and the control algorithm is adapted to fit the platform architecture (as in networked control systems (NCS) where the characteristics of the wireless network such as delay and packet loss probabilities are given and the control algorithms are designed taking them into account), (ii) the control algorithm and its assumptions are given and the platform is designed to meet these assumptions as closely as possible (e.g., by designing appropriate scheduling and resource allocation policies), or (iii) it is a true co-synthesis where the parameters of the control algorithms and the implementation platform are jointly determined within an integrated optimization framework.

1.3.1 Separation of Concerns

Automatic control: It is a well-studied subject with several decades of history and a large pool of design methods. Early works on design and analysis of a control system have focused on the mathematical model of the closed-loop system, including the plant and the controller. The controller is designed such that the system is stable and certain performance requirements are satisfied. In this context, different notions of stability (e.g., stability in the sense of Lyapunov, asymptotic stability, and exponential stability) [95] and several performance metrics (e.g., settling time, peak overshoot, and integral cost) have been defined [96]. Several standard control design techniques are known [97], i.e., (i) pole-placement allows to design stable controllers, (ii) linear quadratic regulator (LQR) optimizes a quadratic cost function on state and control input when the states are fully known, and (iii) linear quadratic Guassian (LQG) control optimizes the cost function using a Kalman filter when all the states are not measurable. However, these

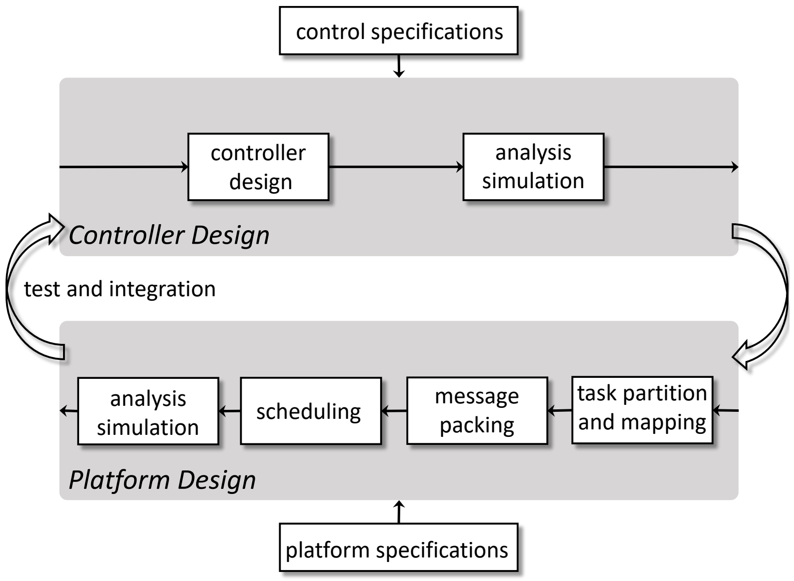


Figure 1.8: Separation of concerns: Control and platform parameters are designed separately followed by integration and testing.

techniques do not directly consider implementation related details like non-negligible and variable times for software execution and data transmission, faulty networks, and finite precision arithmetic.

Platform design for embedded systems: It is also studied extensively and is composed of several stages: (i) task partitioning and mapping, (ii) frame packing (for communication messages), and (iii) task and frame scheduling. Platform design and analysis consider timing properties, e.g., application latencies, periods, relative deadlines, task execution times, and message frame transmission times. The main focus has been to synthesize implementations that are schedulable (i.e., all real-time software tasks meet their deadlines) and resource-efficient (i.e., minimum usage of computation and communication resources). Time-triggered (e.g., OSEK time and FlexRay static segment) and event-triggered implementations (e.g., OSEK and CAN) of software algorithms have been studied in the literature.

In time-triggered implementations, a task is executed or a message is transmitted in an allocated time window. These time windows for all tasks and messages in a system are often calculated by solving a constrained optimization problem [98–101]. Typical constraints include: (i) Processor utilization and bus load can be at most $100\,\%$ or as given by the design specification. (ii) Temoral order of the tasks and messages in an application must be respected, e.g., sensor task must first read the sensor values and then only controller task can calculate the control input according to the control law. (iii) No two tasks or messages mapped on the same resource (processor or bus) must have overlapping time windows.

On the other hand, in event-triggered systems, tasks and messages are dispatched based on events and then they are scheduled based on certain scheduling policies. Typical scheduling policies include fixed-priority preemptive (e.g., OSEK) and non-preemptive (e.g., CAN) scheduling and dynamic-priority scheduling (e.g., earliest deadline first). In fixed-priority scheduling schemes, the main design paramters are the priorities of the tasks and messages.

Typically, priorities are assigned based on well-known heuristics like rate-monotonic and deadline-monotonic scheduling. For event-triggered scheduling policies, there exists different schedulability tests [102, 103] and worst-case response-time analysis techniques [104–108].

Although there exists a rich literature of works for the design and analysis of embedded platforms, the existing theory is not directly applicable to control applications. This is because control requirements like stability and performance cannot always be expressed as timing properties like deadline and period, and however, when expressed in such a form, the parameters can be overly pessimistic.

The semantic gap: In the context of CPSs, the separate design of controllers and platform parameters leads to a semantic gap between the system models considered in the controller design and the actual implementation.

On one hand, the controller design only considers the mathematical model of the physical plant. Therefore, system stability and performance are derived without considering the cyber part, i.e., the implementation on the embedded platform. However, the implementation-related timing properties like sampling period and sensing-to-actuation delay may degrade the performance and in the worst case may also cause system instability. Thus, the semantics of the control models may not be preserved in the implementation when the controller design is oblivious to the implementation details.

On the other hand, the synthesis of platform parameters are based on the software-level timing details and does not consider control-theoretic metrics like stability and performance. An incorrect timing characterization of control properties can result in an inconsistency between models and their implementation. For example, the performance requirement can enforce a strict constraint on application latency which if not correctly modeled may not be satisfied by the implementation.

Hence, it is difficult to design a safe CPS with such a separation of concerns due to the associated semantic gap. We define a CPS to be safe when the corresponding software implementation meets the control requirements on stability and performance even in the worst case. Now, to ensure safety with the separation of concerns, the whole process is usually carried out in an iterative manner as shown in Figure 1.8. Here, the controllers and platform parameters are separately designed followed by integration and testing. In case a test shows that the requirements are not met, the steps are reiterated, possibly without any systematic feedback for improvement. This paradigm relies strongly on the prior experience of engineers and can be error-prone. With the increasing size and complexity of modern embedded systems, this design paradigm is not sustainable. This leads to the need for new design approaches that can guarantee safety in a correct-by-design manner and do not depend on testing.

1.3.2 CPS-Oriented Design Approaches

Due to the strong interplay between controller and platform designs, they are gradually coming closer towards a more CPS-oriented design paradigm.

Platform-aware controller design and analysis: Control theorists have started accounting for implementation details and constraints of the underlying embedded platform and are integrating them in the mathematical models for controller design. In control theory, there exists mature

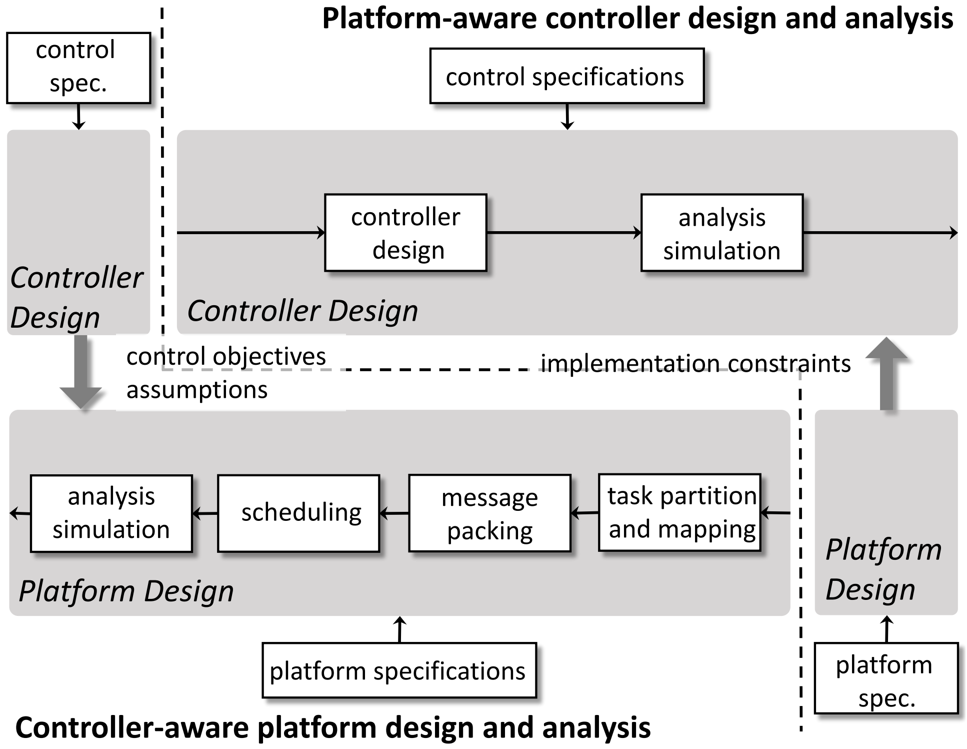


Figure 1.9: CPS-oriented approaches: (i) In platform-aware controller design and analysis, controllers are designed or control properties are analyzed considering the details of the implementation details. (ii) In controller-aware platform design and analysis, platform parameters are designed or analyzed based on controller specification.

design and analysis techniques to handle delayed systems [97] which can be applied to consider fixed sensing-to-actuation delay. Moreover, for multi-output systems, there might be a different delay for each actuation signal that must be considered while determining the control law [109]. However, in event-triggered systems, task executions and message transmission do not occur in precise time windows, and therefore, controller design must consider these timing uncertainties. Towards this, Cervin et al. in [110] has proposed an analysis of system stability and worst-case performance considering both sampling and actuation jitters. On the other hand, [111] has proposed to do proactive delay compensation for improving performance in the presence of large output jitters in wireless multi-hop networks.

Traditionally, controllers are implemented as hard real-time applications using only high-quality resources that offers a certain level of timing determinism. However, this is expensive especially for the cost-sensitive CPS domains. In the last decade, there have been works showing that the inherent robustness in the control loops allows the tasks to miss deadlines and packets to be dropped. In such cases, control loops are modeled as switched systems and are analyzed to derive weakly-hard constraints that must be satisfied during the implementation [112–117]. These constraints are typically expressed as $(m, k)-$firm rule which states that at least m out of k times the control loop must be successfully closed.

Besides timing uncertainties, there are also constraints on the choice of sampling periods. For example, only a discrete set of periods are allowed in OSEK due to a limited number of timers. For such a constraint, the straightforward scheme is to find the largest sampling period from the pre-configured set for which a controller can be designed satisfying the requirements. However, this might result in using an unnecessarily high sampling rate, and thereby overloading the processor. To address this issue, [117, 118] have proposed to design resource-efficient controllers with non-uniform sampling.

Besides the timing details of the implementation, stability and control performance are also influenced by the quantization errors that occur due to the finite-precision arithmetic in processors. Towards this, there have been works on quantization error aware verification of control software [119–122], where the goal is to verify that the final state of the system is within a bounded region of the equilibrium state. Furthermore, Majumdar et al. in [123] have proposed to synthesize controllers by co-optimizing the LQR cost function and the quantization error, thereby constructing a Pareto front of the two objectives.

In summary, platform-level details like sensing-to-actuation delay, sampling and actuation jitter, packet drops, deadline misses, and finite-precision arithmetic are modeled and considered in the controller design phase, so that the designed controllers are platform-aware.

Controller-aware platform design and analysis: In this design paradigm, engineers study properties of control loops and consider them in the design of embedded platforms. As mentioned earlier, control stability and performance are influenced by the choice of platform parameters. Thus, during the platform design, if these control properties are considered as constraints, then it is possible to synthesize CPSs in a correct-by-construction manner. In this context, [124, 125] have suggested techniques to calculate priorities and periods of control tasks for fixed-priority preemptive scheduling in the processor while ensuring stability and maximizing control performance respectively. Similarly, [126] has proposed a heuristic to efficiently compute time-triggered schedules for control tasks and messages. Furthermore, in [127, 128], server based scheduling of control tasks has been studied, where parameters of control servers are synthesized by taking into account the stability and worst-case performance requirements of controllers. While these works consider implementation of controller only, there have been studies that consider a mixed-criticality system settings. In such a setting, control and non-control tasks are considered together. While for non-control tasks, deadlines need to be met, for control tasks, stability and performance need to be guaranteed. Here, Wu et al. in [129] determine periods and deadlines of control tasks for the EDF scheduling policy such that all tasks meet their deadlines while the overall performance of the system is maximized. Schneider et al. in [130, 131] calculate priorities of all tasks in the system for the fixed priority scheduling scheme while maximizing the system performance.

While control theorists have derived $(m, k)-$firm constraints for controllers implemented using unreliable platform resources, these constraints can also be exploited to design resource-efficient implementations. In this context, Majumdar et al. in [116] have proposed a static scheduling algorithm that satisfies the $(m, k)-$firmness of all controllers and at the same time tries to maximize the overall performance of the system. In the same vein, there have been works analyzing or formally verifying $(m.k)-$firmness of controllers for given implementations [4, 132–134].

To minimize resource usage, event- and self-triggered controllers are studied. In event-triggered control, the control law is executed only when a certain error threshold is violated for the current system states [135]. This decision is taken by a feedback scheduler that also monitors the system states. On the other hand, self-triggered controllers calculate the current control input and also the next sampling instant based on the current system states [136]. Therefore, they do not require an additional feedback scheduler. For these control schemes, triggering instants are determined based on different stability and performance requirements [137–139]. Schedulability of event- and self-triggered controllers is also an important design aspect [140, 141].

Towards reducing system cost, researchers have further studied multi-resource platforms consisting of high- and low-quality resources [142, 143]. While high-quality resources offer timing guarantees, they are expensive, and therefore, they must be optimally distributed. In this context, hybrid implementation of controllers have been proposed, where a controller uses low-quality resources when the system is in steady state while it requests to switch to high-quality resources on disturbance [144]. For such implementations, tighter dimensioning of expensive high-quality resources have been considered by appropriate analyses of control requirements [145, 146].

To summarize, the controller implementation (e.g., schedule paramters for tasks and messages) can be tuned according to control objectives like stability and performance, rather than solely on intermediate objectives like deadline and latency.

Safe, yet not optimal: The aforemetioned CPS-oriented approaches, as shown in Figure 1.9, consider realistic platform details while designing a controller or they study control specification for platform implementation. In particular, they mathematically translate control properties into timing characteristics and vice-versa to bridge the semantic gap between controller and platform designs. However, these methods consider the parameters on one side as given and design the parameters on the other side accordingly, and thus, there is a limited opportunity for optimization. They may result in a sub-optimal design configuration with respect to control performance, resource efficiency or both. In order to achieve higher design efficiency, it is important to design the control and the platform parameters together from joint specification in a holistic optimization framework.

1.3.3 Control-Platform Co-Synthesis for CPSs

In the cost-sensitive automotive domain, it is not only necessary to ensure safety but it is equally desirable to achieve design optimality. We emphasize on the importance of control-platform co-synthesis for CPSs towards safe and optimal designs.

In recent years, several control-platform co-synthesis approaches have been proposed that consider the design of control and platform parameters as a holistic optimization, as shown in Figure 1.10. Generally, the co-synthesis problem is formulated as a non-convex optimization problem and is solved using a customized DSE technique. The solutions provide both sets of parameters that are tuned according to certain objectives like control performance and resource usage. Therefore, the synthesized parameters represent an optimal design configuration. Moreover, the control model semantics are fully preserved in the implementation. This is because the controllers are designed according to the detailed constraints from the platform side and the

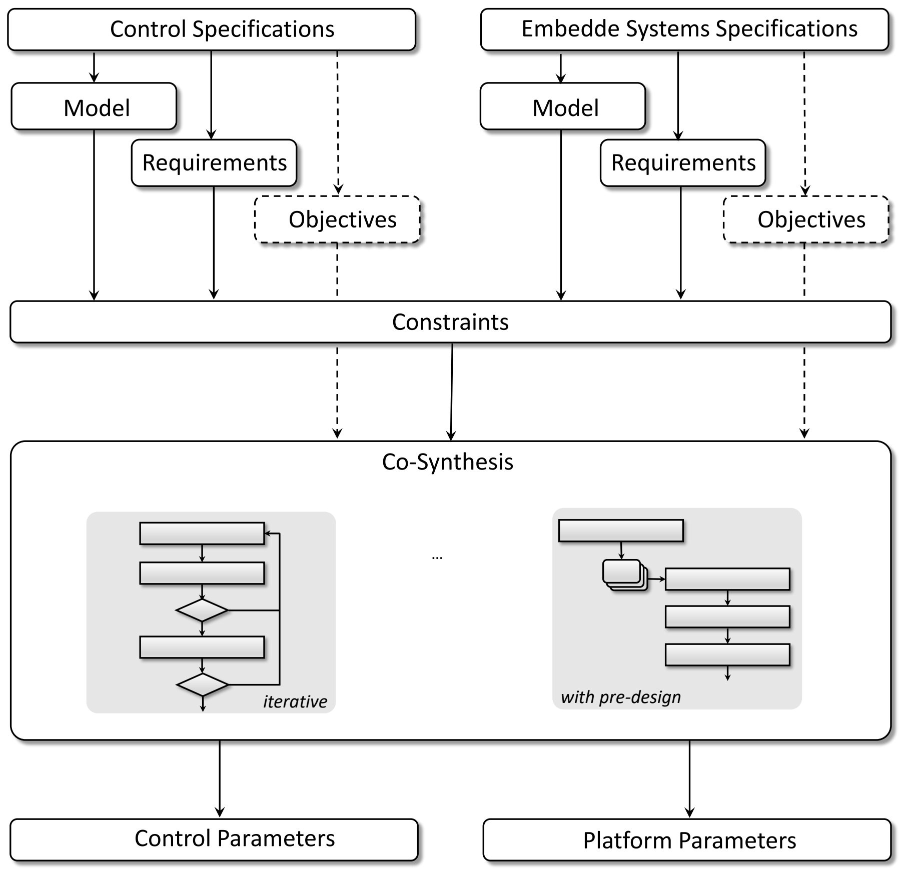


Figure 1.10: Control-platform co-synthesis for CPSs: Control and platform parameters are designed in an integrated optimization framework.

platform parameters are synthesized considering stability and performance requirements from the control side. The synthesized parameters are, therefore, correct-by-design and ensure safety.

Co-synthesis approaches: One of the earliest approaches on integrated controller design and scheduling is proposed by Aminifar et al. in [147]. This approach synthesizes LQG controllers, sampling periods, and scheduling priorities of control tasks, while maximizing the expected control performance and guaranteeing the worst-case performance. Here, an iterative hybrid optimization technique is employed where the outer layer performs search on the periods and the inner layer assigns priorities to the control tasks based on control-theoretic properties. Later, [148] extends [147] to consider controller-server co-design, where controllers and server parameters are designed simultaneously.

For distributed embedded controllers, communication schedules must also be calculated during the co-synthesis. [149] is one of the first few works to propose a control-platform co-synthesis approach for distributed systems, where both static cycling scheduling and priority based scheduling is studied. In these settings, controllers, sampling periods, and static schedules (or priorities) are determined using nested two-layer optimization technique. For priority-based scheduling, genetic algorithm is used in both layers to decide on the periods and priorities respectively, while in case of static scheduling constrained logic programming is used to determine the static schedules in the inner layer. Later, [150] extends this work with specific characterization of the FlexRay parameters. Similarly, Aminifar et al. in [151] have extended the approach developed for single-processor architecture [147], with added complexity due to schedule computation for the CAN bus.

Challenges and future outlook: There exist multiple challenges that need to be addressed, if the aforementioned approaches are to be applied to industrial-scale systems. As described in detail in Section 1.2.2, these challenges include complex and large design space, model integration, appropriate objective formulation, and lack of integrated toolchains. Moreover, existing approaches have not thoroughly considered several aspects of platform architectures, e.g., memory hierarchy [152], heterogeneous networks [153], or multi-core processors [154] – all of which are common in modern embedded systems. Furthermore, they also do not take into account complex characteristics of control systems, e.g., time variance, nonlinearity, or input saturation. Hence, control-platform co-synthesis is a promising research direction with a number of open problems.

1.4 Contributions and Organization

Contributions: In this thesis, we propose correct-by-construction approaches for the design of distributed control systems and cell balancing algorithms respectively. Our works mainly have a CPS-oriented approach and can be categorized under controller-aware design of embedded platforms, platform-aware design of control algorithms, and control-platform co-design respectively. The proposed techniques study integrated models of the systems and therefore, optimize design objectives more comprehensively. These optimization objectives are control performance and resource usage for distributed control systems and energy dissipation and balancing time for cell balancing algorithms. Although our proposed techniques are mainly targerted towards automotive CPS, these approaches can either be applied directly or extended to other CPS settings. For example, distributed control systems are also common in other CPS domains like avionics and industry automation. Similarly, battery packs are used in smart grids as well. Moreover, the charge transfer circuits for cell balancing are analogous to transportation networks. That is, two charge transfers cannot be scheduled simulataneously if their current flow paths overlap, which is similar to the consideration that conflicting traffics cannot enter a road intersection at the same time. Thus, our proposed approach for scheduling charge transfers can be extended to manage traffic intersections and replace existing techniques, e.g., [155]. Nevertheless, in the context of automotive CPSs, we make the following contributions in this thesis.

- We propose a co-optimization approach for a time-triggered distributed CPS comprising multiple control applications. The proposed approach maximizes the average control performance of the system and minimizes the resource usage, while for each application, it determines the control law and the task and message schedules implementating the controller. Note that the two design objectives are often conflicting, i.e., to improve control performance more resources need to be allocated to the controllers. The co-optimization approach consists of two stages. In the first stage, several optimal controllers are predesigned for each application considering all feasible implementations. In the second stage, a customized hybrid optimization technique is used to construct a Pareto front. Here, a Pareto point is a valid design configuration showing a trade-off between the objectives and comprising a set of controllers and their implementation parameters corresponding to the set of applications.

- We also address the challenge of incorporating the aforementioned state-of-the-art co-optimization approach in commercial-off-the-shelf (COTS) tools for the design of automotive systems. We have developed an integrated tool-suite *Co-Flex* for automated design and implementation of distributed control systems. Co-Flex comprises standard tools from MATLAB/Simulink for the modeling and analysis of control systems. On the other hand, Elektrobit's SIMTOOLS is used for the specification, modeling and analysis of the embedded platforms. In addition, Co-Flex offers a library comprising template blocks that can be configured to specify details of a distributed control application, e.g., the plant to be controlled, the control requirements, how the controller is mapped on the platform, among others. Using these information, a tool can be invoked to predesign prospective optimal controllers for the application, i.e., the first stage of the co-optimization. Furthermore, a tool is offered that automatically reads all relevant information provided in the Co-Flex and SIMTOOOLS specification blocks. Based on the extracted specification, a tool runs the hybrid optimization to generate the Pareto front. The systems engineer can select the preferred design configuration, i.e., the Pareto point with the optimal trade-off between average control performance and resource usage. Based on designer's choice, the software model is automatically created using Simulink and SIMTOOLS. From this model, binaries can be generated for all ECUs using Simulink Real-Time Workshop (RTW) and Elektrobit's SIMTARGET. These binaries can then be flashed on to the respective ECUs.

- We synthesize a cost-efficient static communication schedule for a set of distributed control applications mapped on a heterogeneous multi-resource platform. Here, system cost is reduced by sharing expensive high-quality resources among the applications such that each application gets only the minimum required amount of such resources, while for the remaining time the controller uses cheaper low-quality resources. Here, a thorough analysis of the control dynamics is needed for the switching scheduling strategy. Based on the results of the analysis, we formulate a constrained optimization problem to determine the schedules of the applications such that the minimum amount of high-quality resources are used, while guaranteeing that all applications meet their worst-case performance requirements.

- For the same problem setting as above, we also propose a dynamic scheduling strategy for the distributed controllers. According to the proposed strategy, an application uses low-quality resources when the controlled plant is in steady state, and it requests the scheduler to switch to high-quality resources on the arrival of a disturbance. The scheduler arbitrates between requests based on their criticality, which is determined by the time an application can still wait for the high-quality resources without violating its control requirement. Once an application gets the high-quality resources, it uses them at least until a certain minimum performance is guaranteed. For such a dynamic scheduling scheme, we also propose a nested two-layer optimization approach to determine the minimum amount of high-quality resources necessary to meet the requirements of all applications. The approach comprises a first-fit heuristic in the outer layer and model checking of a network of timed automata (TAs) in the inner layer.

- We study a BMS architecture where SoCs of all cells are reported to a master ECU that determines a charge transfer schedule for cell balancing. This enables the master ECU to take a holistic control decision considering the optimization objectives. For this setup, we propose a closed-form optimal approach to minimize the total energy dissipation during cell balancing. We exploit the electrochemical properties of Li-ion cells and appropriately model the charge transfer process considering the underlying control circuits. We show that the total energy dissipation in a charge equalization process depends only on the choice of charge transfers (i.e., pairs of cells and their durations of charge transfer) and not on the exact schedule of the charge transfers. Thus, we can formulate a mixed-integer linear programming (MILP) with charge equalization as the constraint and minimization of energy dissipation as the objective. Solution to the MILP gives the amount of time each feasible pair of cells must be scheduled for charge transfer to realize charge equalization with minimum energy dissipation. Subsequently, these charge transfers are scheduled using an appropriately scheduling algorithm.

- In the context of cell balancing, we also propose an optimal technique to minimize the balancing time for the aforementioned BMS setup. Here, we show that for a given set of charge transfers, we can formulate a minimum vertex coloring (MVC) problem to schedule them in minimum time while respecting the constraints of the underlying charge transfer circuit. Here, we represent the scheduling conflicts between charge transfers in a graph where each vertex is a unit charge transfer time between a pair of cells and an edge between two vertices imply that the corresponding two charge transfers cannot be scheduled at the same time. We further prove that the constructed graph is an interval graph, i.e., the vertices can represent an interval on real line and an edge between two vertices means that the corresponding two intervals overlap. Next, we exploit the properties of the interval graphs and the characteristics of the cell balancing problem to derive a linear expresssion for balancing time in terms of durations of charge transfer between the feasible pairs of cells. Thus, we can use the derived expression as the optimization objective to formulate an MILP that also considers the constraint for charge equalization. Solution to the MILP provides a set of charge transfers which when scheduled using the MVC algorithm for interval graphs will guarantee minimum balancing time. Note that the proposed optimization approach consists of two stages, i.e., the MILP and the MVC, however, the algorithm used in the second stage influences the objective formulation in the first stage.

Organization: The remaining chapters of this thesis provide the necessary mathematical background and describe our proposed techniques and tools. The organization of the rest of this thesis is explained as follows:

- Chapter 2 provides the existing mathematical models for the problem settings studied in this thesis. In particular, it discusses the necessary mathematical background on (i) feedback control systems, (ii) time-triggered software tasks, (iii) FlexRay communication, and (iv) charge transfer circuit architectures for active cell balancing. Furthermore, it briefly explains the well-known optimization techniques that we have employed in our works, i.e., MILP, Satisfiability Modulo Theories (SMT), and MVC.

- Chapter 3 explains the proposed co-optimization approach for distributed control systems that synthesizes controllers and platform paramaters simultaneously, while maximizing the average control performance and minimizing the resource usage.

- Chapter 4 illustrates the proposed tool-suite Co-Flex that integrates existing COTS tools (like MATLAB/Simulink and SIMTOOLS/SIMTARGET) and also offers additional tools to facilitate convenient and automated design and implementation of distributed control systems for an automotive setting.

- Chapter 5 describes the proposed static and dynamic scheduling approaches for distributed control applications that enables tighter dimensioning of the expensive high-quality resources in a heterogeneous multi-resource CPS setting.

- Chapter 6 outlines our efforts in the design of optimal control algorithms for active cell balancing that determine the energy-optimal and the time-optimal schedules respectively for charge transfer between feasible pairs of cells.

- Chapter 7 concludes this thesis with the main takeaway points and provides some promising future directions in the area of modeling, design and analysis of CPSs.

1.5 Bibliographic Notes

The research related to this doctoral dissertation has primarily been in the area of modeling, design, and analysis of cyber-physical systems, which has led to 23 accepted publications. In this section, publications related to this thesis are first listed. Additional publications that are in the same direction, but not related to this thesis, are also listed and interested readers can refer to them to explore further possibilities of multi-domain coupling in CPSs. Among these publications, one has received the Best Paper Award and two have been additionally nominated for the Best Paper Award.

1.5.1 Thesis-Related Publications

Parts of this thesis have appeared in the following publications.

- Parts of Chapter 1 are discussed in the following publications:

  [1] Debayan Roy, Michael Balszun, Thomas Heurung, and Samarjit Chakraborty. *Multi-domain coupling for automated synthesis of distributed cyber-physical systems.* International Symposium on Circuits and Systems (ISCAS), 2018.

  [2] Debayan Roy, Michael Balszun, Thomas Heurung, Samarjit Chakraborty, and Amol Naik. *Waterfall is too slow, let's go Agile: Multi-domain coupling for synthesizing automotive cyber-physical systems.* International Conference on Computer Aided Design (ICCAD), 2018.

[3] Debayan Roy, Licong Zhang, Wanli Chang, Sanjoy Mitter, and Samarjit Chakraborty. *Semantics-preserving co-synthesis of cyber-physical systems.* Proceedings of the IEEE - Special Issue on Safe and Secure CPS/IoT, 2018.

- Chapter 3 is related to the following publications:

[4] Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, and Samarjit Chakraborty. *Multi-objective co-optimization of FlexRay-based distributed control systems.* Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016.

[5] Debayan Roy, Licong Zhang, Wanli Chang, and Samarjit Chakraborty. *Automated synthesis of cyber-physical systems from joint controller/architecture specifications.* Forum on Specification & Design Languages (FDL), 2016.

[6] Wanli Chang, Licong Zhang, Debayan Roy, and Samarjit Chakraborty. *Control/architecture codesign for cyber-physical systems.* A book chapter in the Handbook of Hardware/Software Codesign, Eds: Soonhoi Ha and Jürgen Teich, Springer.

[7] Wanli Chang, Debayan Roy, Licong Zhang, and Samarjit Chakraborty. *Model-based design of resource-efficient automotive control software.* International Conference on Computer Aided Design (ICCAD), 2016.

- Chapter 4 will appear in the following publication:

[8] Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, Birgit Vogel-Heuser, and Samarjit Chakraborty. *Tool integration for automated synthesis of distributed embedded controllers.* ACM Transactions on Cyber-Physical Systems (TCPS), 2021. (to appear)

- Chapter 5 is related to the following publications:

[9] Debayan Roy, Wanli Chang, Sanjoy Mitter, and Samarjit Chakraborty. *Tighter dimensioning of heterogeneous multi-resource autonomous CPS with control performance guarantees.* Design Automation Conference (DAC), 2019.

[10] Debayan Roy, Sumana Ghosh, Marco Caccamo, and Samarjit Chakraborty. *Good-Spread: Criticality-aware static scheduling of CPS with multi-QoS resources.* Real-Time Systems Symposium (RTSS), 2020.

[11] Michael Balszun, Debayan Roy, Licong Zhang, Wanli Chang, and Samarjit Chakraborty. *Effectively utilizing elastic resources in networked control systems.* International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017. **[Best Paper Award]**

[12] Debayan Roy, Michael Balszun, Dip Goswami, and Samarjit Chakraborty. *Hybrid automotive in-vehicle networks.* International Symposium on Networks-on-Chip (NOCS), 2017.

[13] Leslie Maldonado, Wanli Chang, Debayan Roy, Anuradha Annaswamy, Dip Goswami, and Samarjit Chakraborty. *Exploiting system dynamics for resource-efficient automotive CPS design.* Design, Automation and Test in Europe (DATE), 2019. **[Best Paper Nomination]**

- Chapter 6 has appeared in the following publications:

[14] Debayan Roy, Swaminathan Narayanaswamy, Alma Pröbstl, and Samarjit Chakraborty. *Multi-stage optimization for energy-efficient active cell balancing in battery packs*. International Conference on Computer Aided Design (ICCAD), 2019.

[15] Debayan Roy, Swaminathan Narayanaswamy, Alma Pröbstl, and Samarjit Chakraborty. *Optimal scheduling for active cell balancing*. Real-Time Systems Symposium (RTSS), 2019.

1.5.2 Other Publications

The following publications are also towards efficient design of CPSs, however, they are not part of this thesis.

[16] Licong Zhang, Debayan Roy, Philipp Mundhenk, and Samarjit Chakraborty. *Schedule management framework for cloud-based future automotive software systems*. International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016.

[17] Ramesh S, Birgit Vogel-Heuser, Wanli Chang, Debayan Roy, Licong Zhang, and Samarjit Chakraborty. *Specification, verification and design of evolving automotive software*. Design Automation Conference (DAC), 2017.

[18] Philipp Mundhenk, Ghizlane Tibba, Licong Zhang, Felix Reimann, Debayan Roy, and Samarjit Chakraborty. *Dynamic platforms for uncertainty management in future automotive E/E architectures*. Design Automation Conference (DAC), 2017.

[19] Qi Zhu, Hengyi Liang, Licong Zhang, Debayan Roy, Wenchao Li, and Samarjit Chakraborty. *Extensibility-driven automotive in-vehicle architecture design*. Design Automation Conference (DAC), 2017.

[20] Wanli Chang, Debayan Roy, Xiaobo Sharon Hu, and Samarjit Chakraborty. *Cache-aware task scheduling for maximizing control performance*. Design, Automation and Test in Europe (DATE), 2018.

[21] Hengyi Liang, Zhilu Wang, Debayan Roy, Soumyajit Dey, Samarjit Chakraborty, and Qi Zhu. *Security-driven codesign with weakly-hard constraints*. International Conference on Computer Design (ICCD), 2019.

[22] Wanli Chang, Debayan Roy, Shuai Zhao, Anuradha Annaswamy, and Samarjit Chakraborty. *CPS-oriented modeling and control of traffic signals using adaptive back pressure*. Design, Automation and Test in Europe (DATE), 2020. **[Best Paper Nomination]**

[23] Anna Minaeva, Debayan Roy, Benny Akesson, Zdenek Hanzálek, and Samarjit Chakraborty. *Efficient heuristic approach for control performance optimization in time-triggered periodic scheduling*. IEEE Transactions on Computers, 2020.

2 Mathematical Background

In this thesis, we study existing models from different design domains of CPSs and try to mathematically derive the interplay between them. We formulate design optimization problems using these models and by properly characterizing the interplay variables. These problems are then solved by combining different well-established optimization techniques depending on the characteristics of the problems. Towards better understanding of our proposed design approaches, this chapter provides the necessary mathematical background.

Chapter Organization: This chapter is organized in four sections. Section 2.1 introduces feedback control systems and derives the equivalent discrete-time mathematical models for delayed systems from the continuous-time models. Furthermore, it states the stability condition for such systems and explains a controller design technique that guarantees asymptotic stability. In Section 2.2, we discuss time-triggered scheduling of tasks and provide the timing models of time-triggered tasks. Subsequently, we summarize relevant details of FlexRay protocol and model the two scheduling schemes supported by FlexRay. Section 2.3 explains the working principle of two different charge transfer architectures and derives the mathematical models of charge transfer between two cells using these architectures. In Section 2.4, we outline different optimization and verification models used in this thesis. In particular, linear programming (LP), integer linear programming (ILP), mixed-integer linear programming (MILP), particle swarm optimization (PSO), Satisfiability Modulo Theories (SMT), timed automata (TAs), and minimum vertex coloring (MVC) are discussed with examples.

2.1 Feedback Control Systems

Control systems form an integral part of technological advancement in almost every field. We may say that control systems help to derive intended functionality from machines and make processes run adapting to the environment variables. More often than not, control systems

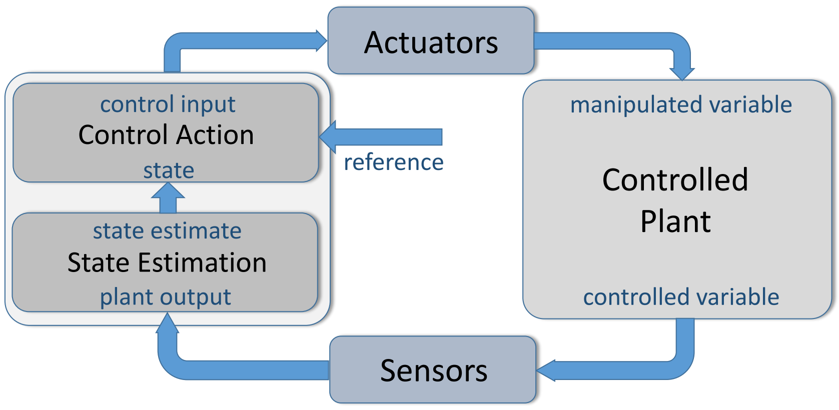


Figure 2.1: Block diagram representing feedback control systems. In such systems, the control action is taken based on the states of the system. When all states are not measurable, it is required to estimate them.

are based on the theory of feedback as depicted in Figure 2.1. In feedback control systems, control action is decided based on (i) the values of the state variables of the controlled plant and (ii) the reference that the plant must follow. In practice, some variables of the plant may not be measurable, and therefore, the corresponding values are estimated using an estimator. The basic idea is to mitigate the error between the plant output and the reference and correspondingly manipulate the plant satisfying some high-level requirements on safety and performance.

2.1.1 Continuous-Time State-Space Model

In this thesis, we study linear and time-invariant (LTI) feedback control systems. The state-space mathematical model of the dynamic behaviour of such a system in continuous time can be represented by the following differential equations:

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad y(t) = Cx(t), \tag{2.1}$$

where the vectors $x(t) \in \mathbb{R}^{n\times 1}$, $y(t) \in \mathbb{R}^{p\times 1}$, and $u(t) \in \mathbb{R}^{m\times 1}$ represent the system states, the system output, and the control input respectively at time instant t[1]. Here, the constant matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$ are respectively the state, the input and the output matrices.

Example: DC Motor Speed Control System

- The system dynamics of a DC motor speed control system are given by the following differential equations:

$$J_m\ddot{\theta}_m + b_m\dot{\theta}_m = K_{t,m}i_m,$$
$$L_m\dot{i}_m + R_m i_m = V_m - K_{b,m}\dot{\theta}_m,$$

[1]$\mathbb{R}$ represents the set of real numbers.

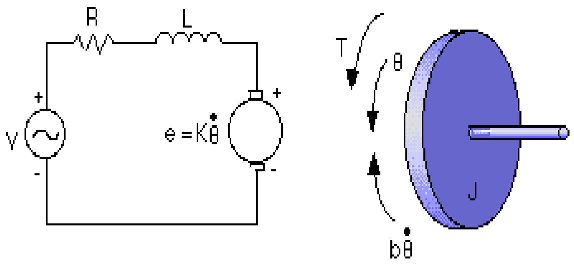


Figure 2.2: DC motor speed control system. The voltage across the motor armature can be varied to control the motor angular speed.

where, (i) the constants are as follows: J_m is the moment of inertia of the rotor, b_m is the motor viscous friction constant, $K_{t,m}$ is the motor torque constant, L_m and R_m are the electric inductance and resistance respectively of the motor armature circuit, and $K_{b,m}$ is the electromotive force constant; and (ii) the continuous variables are as follows: θ_m is the motor angular position, i_m is the motor current, and V_m is the voltage applied at the motor armature.

- In a DC motor speed control system, we need to regulate the speed. Thus the system output is given by:
$$y = \dot{\theta}_m.$$
- Speed can be controlled by manipulating the voltage at the motor armature. Thus, the control input is given by:
$$u = V_m.$$
- By studying the system dynamics given by the differential equations, we can determine the state vector as follows:
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_m \\ i_m \end{bmatrix}.$$
- The state-space model is, therefore, given by:
$$\begin{bmatrix} \ddot{\theta}_m = -\frac{b_m}{J_m}\dot{\theta}_m + \frac{K_{t,m}}{J_m} i_m \\ \dot{i}_m = -\frac{K_{b,m}}{L_m}\dot{\theta}_m - \frac{R_m}{L_m} i + \frac{1}{L} V \end{bmatrix} \Longleftrightarrow \dot{x} = \begin{bmatrix} -\frac{b_m}{J_m} & \frac{K_{t,m}}{J_m} \\ -\frac{K_{b,m}}{L_m} & -\frac{R_m}{L_m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L_m} \end{bmatrix} u$$
$$y = \dot{\theta}_m \Longleftrightarrow y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$
- The constant matrices are, therefore, given by:
$$A = \begin{bmatrix} -\frac{b_m}{J_m} & \frac{K_{t,m}}{J_m} \\ -\frac{K_{b,m}}{L_m} & -\frac{R_m}{L_m} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{L_m} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{2.2}$$

Solution to the differential equation: Given a continuous-time state-space model as in Eq. (2.1), we can solve the first-order differential equation to determine an expression for the states at time t. Towards this, we take a Laplace transform [156] of Eq. (2.1) and get,

$$\begin{aligned} & sX(s) - x(0) = AX(s) + BU(s) \\ \Longrightarrow & (sI - A)X(s) = x(0) + BU(s) \\ \Longrightarrow & X(s) = (sI - A)^{-1}x(0) + (sI - A)^{-1}BU(s). \end{aligned} \tag{2.3}$$

Now, taking inverse Laplace of Eq. (2.3),

$$\mathcal{L}^{-1}[X(s)] = \mathcal{L}^{-1}[(sI - A)^{-1}]x(0) + \mathcal{L}^{-1}[(sI - A)^{-1}BU(s)],$$

we get,

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau. \tag{2.4}$$

2.1.2 Discrete-Time State-Space Model

Considering the fact that the controller is implemented on an embedded platform with limited resources, the control input is applied to the plant only at discrete instants t_k where $k \in \mathbb{Z}^*$.[2] Let us assume that the time interval between two consecutive instants is a constant h and the control input to the plant is held constant till the next input is generated and applied, i.e., $u(t) = u(t_k)$, where $t_k \leq t < t_{k+1}$. This is equivalent to a system with sample and hold device connected at the input, and correspondingly, h is the sampling period of the system.

The equivalent state-space model of the sampled-data (discrete-time) control system [97] can be represented by the following difference equations:

$$x[k+1] = \phi x[k] + \Gamma u[k], \qquad y[k] = Cx[k],^3 \tag{2.5}$$

where the discrete-time state and input matrices ϕ and Γ can be derived from the continuous-time matrices for a given sampling period h as follows:

$$\phi = e^{Ah}, \quad \Gamma = \int_0^h (e^{A\tau'}d\tau') \cdot B. \tag{2.6}$$

Proving Eqs. (2.5) and (2.6): From Eq. (2.4), we can express $x(t_k)$ and $x(t_{k+1})$ as follows:

$$x(t_k) = e^{At_k}x(0) + \int_0^{t_k} e^{A(t_k-\tau)}Bu(\tau)d\tau. \tag{2.7}$$

[2] $\mathbb{Z}^*$ represents the set of non-negative integers.

[3] $x[k]$ and $u[k]$ represents the system states and control input at the time instant t_k, i.e., $x[k] = x(t_k)$ and $u[k] = u(t_k)$.

$$x(t_{k+1}) = e^{At_{k+1}}x(0) + \int_0^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \quad . \tag{2.8}$$

Eq. (2.8) can be rewritten as follows:

$$\begin{aligned}x(t_{k+1}) &= e^{A(t_{k+1}-t_k)}e^{At_k}x(0) + \int_0^{t_k} e^{A(t_{k+1}-t_k)}e^{A(t_k-\tau)}Bu(\tau)d\tau + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \\ &= e^{A(t_{k+1}-t_k)}\Big[e^{At_k}x(0) + \int_0^{t_k} e^{A(t_k-\tau)}Bu(\tau)d\tau\Big] + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \\ &= e^{A(t_{k+1}-t_k)}x(t_k) + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \quad \Big[\because Eq.\ (2.7)\Big].\end{aligned}$$

Applying zero order hold (ZOH) to the control input $u(\tau)$ between t_k and t_{k+1}, we get:

$$x(t_{k+1}) = e^{A(t_{k+1}-t_k)}x(t_k) + \Big[\int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bd\tau\Big]u(t_k). \tag{2.9}$$

Changing the integration variable as $\tau' = t_{k+1} - \tau$, we get

$$x(t_{k+1}) = e^{A(t_{k+1}-t_k)}x(t_k) + \Big[\int_0^{t_{k+1}-t_k} e^{A\tau'}Bd\tau'\Big]u(t_k). \tag{2.10}$$

Substituting $t_{k+1} - t_k = h$, we get

$$x(t_{k+1}) = e^{Ah}x(t_k) + \Big[\int_0^{h} e^{A\tau'}Bd\tau'\Big]u(t_k). \tag{2.11}$$

Equivalently, Eq. (2.11) can be written using Eqs. (2.5) and (2.6). Hence, proved.

2.1.3 Delayed Discrete-Time State-Space Model

Ideal discrete-time system model, given by Eq. (2.5), considers zero delay between sensing and actuation. This essentially means that the control input is applied to the plant immediately after the sensors read the plant states. This assumption is idealistic because of two reasons: (i) Computation of control input by a software task takes non-negligible time on an embedded platform. This is because embedded processors often have limited computation bandwidth. (ii) Sensors and actuators are often spatially distributed and control loops may involve some communication over shared network. This might also introduce a significant delay between sensing and actuation.

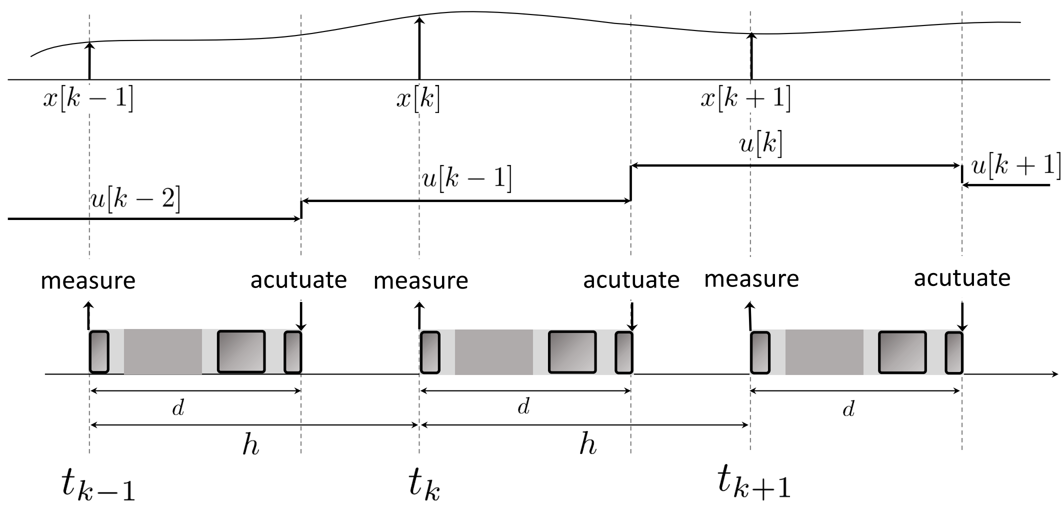


Figure 2.3: Timing diagram of a delayed discrete-time system. The system states are measured periodically every h time units where h is the sampling period. There is also a delay d between sensing and actuation. Thus, between two sampling instants, first the previous control input $u[k-1]$ is applied till d time units and thereafter the new control input $u[k]$ is applied for the remaining time $h-d$.

In reality, the timings of a discrete-time system are as shown in Figure 2.3. Corresponding to the figure, we may write $u(t) = u[k-1]$ for $t_k \leq t < t_k + d$ and $u(t) = u[k]$ for $t_k + d \leq t < t_{k+1}$, where d is the delay between sensing and actuation. Thus, we rewrite Eq. (2.9) as

$$x[k+1] = e^{A(t_{k+1}-t_k)}x[k] + \Big[\int_{t_k}^{t_k+d} e^{A(t_{k+1}-\tau)}Bd\tau\Big]u[k-1] + \Big[\int_{t_k+d}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bd\tau\Big]u[k]. \quad (2.12)$$

Substituting $\tau' = t_{k+1} - \tau$, we get:

$$\begin{aligned} x[k+1] &= e^{A(t_{k+1}-t_k)}x[k] + \Big[-\int_{h}^{h-d} e^{A\tau'}Bd\tau'\Big]u[k-1] + \Big[-\int_{h-d}^{0} e^{A\tau'}Bd\tau'\Big]u[k] \\ &= e^{Ah}x[k] + \Big[\int_{h-d}^{h} e^{A\tau'}Bd\tau'\Big]u[k-1] + \Big[\int_{0}^{h-d} e^{A\tau'}Bd\tau'\Big]u[k] \\ &\qquad \Big[\because t_{k+1} - t_k = h, \text{ changing the limits of the integration}\Big]. \end{aligned} \quad (2.13)$$

We may rewrite Eq. (2.13) as

$$x[k+1] = \phi x[k] + \Gamma_1 u[k-1] + \Gamma_0 u[k], \quad (2.14)$$

where,

$$\phi = e^{Ah}, \quad \Gamma_0 = \int_{0}^{h-d} e^{A\tau'}Bd\tau', \quad \Gamma_1 = \int_{h-d}^{h} e^{A\tau'}Bd\tau'. \quad (2.15)$$

Eqs. (2.14) and (2.15) give the mathematical model for delayed discrete-time systems [97].

Now, let us consider an augmented state vector $x_a[k] = \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix}$ and rewrite the delayed discrete-time system model [157] as follows:

$$\begin{bmatrix} x[k+1] \\ u[k] \end{bmatrix} = \begin{bmatrix} \phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix} + \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix} u[k], \Longleftrightarrow x_a[k+1] = \phi_a x_a[k] + \Gamma_a u[k],$$
$$y[k] = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix} \Longleftrightarrow y[k] = C_a x_a[k], \tag{2.16}$$

where

$$\phi_a = \begin{bmatrix} \phi & \Gamma_1 \\ 0 & 0 \end{bmatrix}, \quad \Gamma_a = \begin{bmatrix} \Gamma_0 \\ I \end{bmatrix}, \quad C_a = \begin{bmatrix} C & 0 \end{bmatrix}. \tag{2.17}$$

With state augmentation, the state-space model of a delayed discrete-time system is of the same form as Eq. (2.5) and standard techniques like pole-placement and LQR can be applied to design the controller.

2.1.4 Stability of Discrete-Time Systems

System stability is the most essential requirement for designing a controller. To understand the notion of stability, we first define an *equilibrium point* x_e in the state space as a point at which a system remains in the absence of external inputs and/or disturbances. This implies $x[k+1] = x[k]$ when $x[k] = x_e$ and $u[k] = 0$ [95].

An equilibrium point x_e is *stable in the sense of Lyapunov* if for any given value of a parameter $\epsilon > 0$ there exists a number δ such that when $||x[0] - x_e|| < \delta$, then the state vector satisfies $||x[k] - x_e|| < \epsilon$ for all $k > 0$ [95]. That is, when the state of the system is initially within a radius δ from the equilibrium point, it will always remain within a radius ϵ from the equilibrium point in the absence of external inputs or distrubances. This can be formulated as follows:

$$\underset{\epsilon \in \mathbb{R}^+}{\forall} \; \underset{\delta \in \mathbb{R}^+}{\exists} \; \underset{k \in \mathbb{Z}^+}{\forall} (||x[0] - x_e|| < \delta) \implies (||x[k] - x_e|| < \epsilon). \tag{2.18}$$

In this thesis, we consider asymptotic stability of feedback control systems. An equilibrium point x_e is *asymptotically stable* if (i) it is stable in the sense of Lyapunov and (ii) there exists a number $\delta^* > 0$ such that if $||x[0] - x_e|| < \delta^*$, the state vector satisfies $\lim_{k\to\infty} ||x[k] - x_e|| = 0$. Thus, in case of asymptotic stability, the state returns to the equilibrium point eventually if it is initially at a pont within a radius $\delta^* > 0$ from the equilibrium point [95]. This is written as follows:

$$\underset{\delta^* \in \mathbb{R}^+}{\exists} (||x[0] - x_e|| < \delta^*) \implies \lim_{k\to\infty} ||x[k] - x_e|| = 0. \tag{2.19}$$

Condition for asymptotic stability: For an unforced LTI system (i.e., $u[k] = 0 \; \forall k \in \mathbb{Z}^*$) represented as $x[k+1] = \phi x[k]$, given the initial state $x[0]$, we can write

$$x[k] = \phi^k x[0]. \tag{2.20}$$

Without loss of generality, we assume $x_e = \mathbf{0}$. Therefore, for a system to be asymptotically stable with a finite non-zero initial state,

$$\lim_{k \to \infty} ||\phi^k|| = 0. \tag{2.21}$$

This is only possible when the eigenvalues of the of ϕ, i.e., λ_i, $\forall i = 1, 2, .., n$, satisfy

$$|\lambda_i| < 1. \tag{2.22}$$

Here, λ_i also represent a system pole. Thus, for a system to be asymptotically stable, the poles must lie within the unit circle in a complex z-plane [97].

Closed-loop stability of delayed systems: In this thesis, we consider that a control loop is closed using a feedback controller based on the following control law:

$$u[k] = -Kx_a[k] + Fr, \tag{2.23}$$

where r is the reference that the system must track and K and F are the feedback and feedforward gains of the controller.

Corresponding to Eq. (2.23), we can rewrite Eq. (2.16) as

$$\begin{aligned} x_a[k+1] &= (\phi_a - \Gamma_a K)x_a[k] + \Gamma_a F r, \\ y[k] &= C_a x_a[k]. \end{aligned} \tag{2.24}$$

Without loss of generality, we assume $r = 0$. We denote $\phi_{cl} = \phi_a - \Gamma_a K$ and rewrite Eq. (2.24) as follows:

$$x_a[k+1] = \phi_{cl} x_a[k]. \tag{2.25}$$

Therefore, for the closed-loop system to be asymptotically stable the eigenvalues of ϕ_{cl} must lie within the unit circle.

2.1.5 Controller Design

In this thesis, we design controllers ensuring that the closed-loop poles are within the unit circle in the complex z-plane. The controller design problem mainly boils down to determining the values of K and F in the control law (given by Eq. 2.23) such that the closed-loop poles are at desired locations. For a single-input single-output (SISO) LTI system, one can use Ackermann's formula for pole placement [97]. Using Ackermann's formula, the feedback gain K can be calculated as follows:

– Calculate the controllability matrix γ as follows:

$$\gamma = \begin{bmatrix} \Gamma_a & \phi_a \Gamma_a & \cdots & \phi_a^{n-1} \Gamma_a \end{bmatrix}, \tag{2.26}$$

where, n is the number of states in the augmented system model. Check if γ is invertible, i.e., the closed-loop system is controllable. If the closed-loop system is not controllable, then we cannot apply Ackermann's formula.

– Given the set of n poles $\begin{bmatrix}\alpha_1 & \alpha_2 & \cdots \alpha_n\end{bmatrix}$, evaluate the desired characteristic polynomial $H(s)$ of the closed-loop system at ϕ_a as follows:

$$H(\phi_a) = (\phi_a - \alpha_1\mathbf{I})(\phi_a - \alpha_2\mathbf{I})\cdots(\phi_a - \alpha_n\mathbf{I}). \tag{2.27}$$

– Apply Ackermann's formula and calculate K as follows:

$$K = \begin{bmatrix}0 & 0 & \cdots & 1\end{bmatrix}\gamma^{-1}H(\phi_a). \tag{2.28}$$

Given the value of the feedback gain K, we need to determine the value of the feedforward gain F such that the system output follows a reference input. In particular, considering asymptotic stability, the output must satisfy the following constraint:

$$\lim_{k\to\infty} y[k] = r. \tag{2.29}$$

Towards deriving an expression for F, we first take $\mathcal{Z}$-transform [156] of the closed-loop system model in Eq. (2.24) and we get:

$$\begin{aligned} zX_a(z) &= (\phi_a - \Gamma_a K)X_a(z) + \Gamma_a F\frac{z}{z-1}r,\\ Y(z) &= C_a X_a(z). \end{aligned} \tag{2.30}$$

We can further write:

$$\begin{aligned} (zI - \phi_a + \Gamma_a K)X_a(z) &= \frac{z}{z-1}\Gamma_a F r,\\ \implies X_a(z) &= \frac{z}{z-1}(zI - \phi_a + \Gamma_a K)^{-1}\Gamma_a F r. \end{aligned} \tag{2.31}$$

Substituting $X_a(z) = C_a^{-1}Y(z)$, we get:

$$Y(z) = \frac{z}{z-1}C_a(zI - \phi_a + \Gamma_a K)^{-1}\Gamma_a F r. \tag{2.32}$$

Now, applying final value theorem [158], we get:

$$\begin{aligned} \lim_{z\to1}(z-1)Y(z) &= \lim_{z\to1}(z-1)\frac{z}{z-1}C_a(zI - \phi_a + \Gamma_a K)^{-1}\Gamma_a F r\\ &= C_a(I - \phi_a + \Gamma_a K)^{-1}\Gamma_a F r. \end{aligned} \tag{2.33}$$

Considering that the output must reach the reference at infinite time, we get:

$$F = \frac{1}{C_a(I - \phi_a + \Gamma_a K)^{-1}\Gamma_a}. \tag{2.34}$$

Thus, a controller can be designed, i.e., the values of K and F can be calculated, such that the closed-loop system is stable and the system output track the reference.

2.2 Distributed Controller Implementation

Typically, a distributed embedded implementation of a controller comprises several software tasks. A task is a software component or a code snippet that performs a specific function, e.g., reads the values of sensor, executes the control law to calculate the control input, or sends an appropriate control input to the actuator to be applied to the plant. These tasks can be sometimes very complex. For example, when the sensor is a camera, a task might be required to perform non-trivial image processing to obtain useful information for the controller. Moreover, for model-predictive control (MPC) [159], the controller task needs to solve an optimization problem.

Tasks implementing the controller are mapped on different ECUs in a distributed controller implementation, primarily due to the spatial distribution of sensors and actuators. In such a case, the data output of one task might be an input to other task(s). Thus, data needs to be sent from one ECU to another. Data communication is typically realized using messages sent over the communication network. When an application is implemented within a bus cluster, data needs to be sent only over a communication bus (such as FlexRay or CAN) or a series of switches and data links (in case of Automotive Ethernet). However, modern ADAS applications are implemented across bus clusters where data needs to be sent over different buses and communication gateways [153].

An ECU executes software tasks based on a scheduling policy. Similarly, a message is sent over a bus according to the communication protocol of the bus. Thus, the timing models of tasks and messages of a controller implementation are specific to the scheduling scheme used in the processors and the communication protocol of the bus respectively. As mentioned earlier, these implementation-level timings of the tasks and messages influence the performance of the controller. Thus, we study the timing models of tasks and messages for the problem setting considered in this thesis. In particular, we study time-triggered scheduling for tasks and the FlexRay protocol for data communication.

2.2.1 Time-Triggered Tasks

In this thesis, we study time-triggered static scheduling of tasks. According to this scheduling policy, processor time allocation is pre-configured, i.e., it is known when a task will be executed by the ECU [160]. Thus, a time-triggered schedule of a task T_i is defined using a tuple $\{o_i, p_i, e_i\}$, where o_i is the offset, p_i is the period and e_i is the execution time [161]. *Task offset* o_i is defined as the time when the task is triggered to run for the first time. *Task period* p_i is the time difference between two consecutive starts of the task. Typically, the execution time of a software task is usually not deterministic as it depends on several factors such as the kinds of memory accesses made (i.e., cache hits or misses) or the program branches taken (if any) [160]. Thus, we assume that a time window of length e_i equal to the *worst-case execution time* (WCET) is reserved for the task. Thus, in all possible scenarios, the reserved computation time will be sufficient to run the task completely. We assume that the WCET of a task is given by the specification. WCET analysis or calculation is out of scope of this thesis, which is also an important research direction in the context of real-time embedded systems. [162] provides a survey on WCET analysis.

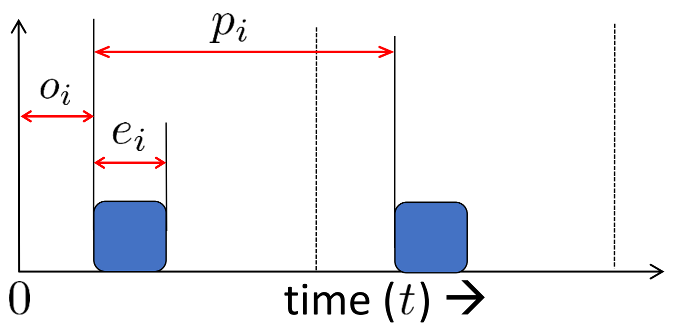


Figure 2.4: Time-triggered scheduling of a software task. Here, a time window of e_i time units is reserved periodically for the task. The first time window starts at o_i time units.

Given a schedule $\{o_i, p_i, e_i\}$ of the task T_i, the timing diagram for the task is as shown in Figure 2.4. Here, T_i runs periodically based on its schedule. The start time of the k-th instance of the task is denoted as $\widehat{t}(T_i, k)$ and is given by:

$$\widehat{t}(T_i, k) = o_i + (k-1) \cdot p_i. \tag{2.35}$$

Thus, the start times of the task instances can be computed as $\{o_i, o_i + p_i, o_i + 2p_i, o_i + 3p_i \cdots\}$. Corresponding to the start time of the k-th task instance, we can also compute the latest finish time of the instance $\widetilde{t}(T_i, k)$ as follows:

$$\widetilde{t}(T_i, k) = \widehat{t}(T_i, k) + e_i = o_i + (k-1) \cdot p_i + e_i. \tag{2.36}$$

Thus, in the worst case, a task instance will finish e_i time units after its start. The aforementioned timing model of time-triggered tasks is later used to formulate constraints for the schedule synthesis of control applications.

2.2.2 FlexRay Communication

As discussed in Section 1.1.2, FlexRay supports hybrid communication protocol, and correspondingly, each FlexRay bus cycle is mainly partitioned into *static* (ST) and *dynamic* (DYN) segments [20, 21], as shown in Figure 2.5. The static segment exhibits TDMA communication and comprises N_s number of *slots* of equal length (Δ), which can be represented as $\mathcal{S}_{ST} = \{1, 2, ..., N_s\}$. Here, a message assigned to a static slot is transmitted within the corresponding time window. Thus, the start and the finish times of a message transmission are known. On the other hand, dynamic segment is partitioned into N_d number of *minislots* of equal length (δ), where typically $\delta << \Delta$. A message assigned to the dynamic segment may consume more than one minislot, as shown in Figure 2.5. A dynamic slot is a logical entity with one or more minislots enabling FTDMA communication. Here, dynamic slots are denoted by $\mathcal{S}_{DYN} = \{N_s + 1, N_s + 2, \cdots\}$.

The hybrid communication is realized using a slot counter, C_s, where the counter starts from 1 at the beginning of each cycle. Here, when $C_s = j$, then the message m_i assigned to the j-th slot is sent over the bus (if ready). In the static segment, the counter is incremented

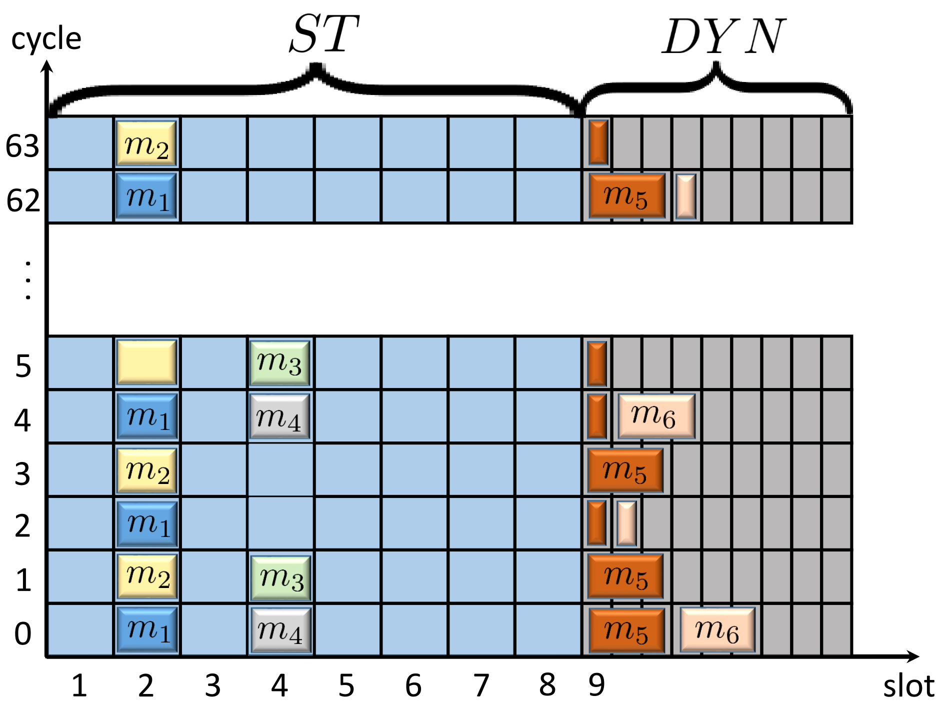


Figure 2.5: An example of FlexRay communication schedule. Four messages are mapped on the static segment while two messages are mapped on the dynamic segment. Static segment enables time-division multiple access (TDMA) communication and the dynamic segment exhibits flexible TDMA (FTDMA) communication. FlexRay 3.0.1 allows slot multiplexing, e.g., m_1 and m_3 are assigned the same slot in different cycles. When there is no data, an entire slot is wasted in the static segment (e.g., slot 2 in cycle 5), while only a minislot is wasted in the dynamic segment (e.g., slots 9 and 10 in cycle 2). While the timings of a message is exactly known in the static segment, there might be communication jitters when using the dynamic segment (e.g., m_6).

after every Δ time units, i.e., at the end of each slot. If no new data has arrived at the beginning of the slot then the whole slot length, i.e., Δ time units, is wasted as shown in Figure 2.5 for m_2 in cycle 5. On the other hand, in the dynamic segment, when a message m_i has some data to be sent then the counter is updated at the end of the last minislot where the message is transmitted. However, if there is no data then the counter is updated at the end of the current minislot, and correspondingly, only δ time units are wasted (refer to Figure 2.5 for messages m_5 and m_6). Thus, the start and the finsh times of a message transmission in the dynamic segment may vary depending on the preceding message transmissions. It may be noted that FlexRay is time-deterministic in the static segment and resource-efficient in the dynamic segment.

FlexRay schedules: FlexRay communication is organized as an infinite repetition of 2^n bus cycles where n is configurable. Each bus cycle is of length T_{bus} time units. Let us consider $n = 6$, i.e., 64 bus cycles which are repeated periodically. Thus, the cycle counter counts from 0 to 63 and then resets. Therefore, the resource allocation in any 64 consecutive cycles will be repeated in the next 64 cycles and so on. In FlexRay communication, a message m_i is transmitted with a schedule that is represented as a tuple $\{s_i, b_i, r_i\}$. These scheduling parameters are described as follows:

- The assigned *slot number* is given by s_i.
- The *base cycle*, b_i, represents the first cycle where the message is allocated the s_i-th slot.
- The *cycle repetition rate*, r_i, denotes the number of cycles between two consecutive time slot allocations.

In Figure 2.5, schedules of the messages are given by $m_1 \equiv \{2, 0, 2\}$, $m_2 \equiv \{2, 1, 2\}$, $m_3 \equiv \{4, 1, 4\}$, $m_4 \equiv \{4, 0, 4\}$, $m_5 \equiv \{9, 0, 1\}$ and $m_6 \equiv \{10, 0, 2\}$.

It must be further noted that FlexRay 3.0.1 [21] allows *slot-multiplexing*, i.e., the same slot number can be assigned to more than one message, however, they must not overlap. This is illustrated by $\{m_1, m_2\}$ and $\{m_3, m_4\}$ in Figure 2.5. In case of FlexRay 2.1, messages can be multiplexed for one slot but to a limted extent, i.e., a slot is assigned to an ECU and the ECU can send different messages in different cycles using the same slot number. For example, in Figure 2.5, if m_1 and m_2 are sent by the same ECU then it is allowed otherwise you get an error in the FlexRay configuration tool.

In addition, there are certain scheduling constraints for FlexRay messages as follows:

- The repetition rates can assume only certain discrete values as follows:
$$r_i \in \{2^k | 0 \leq k \leq n\}. \tag{2.37}$$
This ensures that a schedule computed for 64 cycles when repeated will not violate the periodicity of the messages.
- One slot must be assigned every r_i number of cycles. This requires the following constraint:
$$b_i < r_i.^4 \tag{2.38}$$
This will ensure that there is a slot assigned in the first r_i cycles. Based on the definition of repetition rate, a slot will be assigned every r_i cycles thereafter.

The timing models corresponding to FlexRay messages are provided in [161] that are also used in this thesis. For a message m_i mapped on the static segment, the start time of the k-th instance of the message, which is denoted as $\widehat{t}(m_i, k)$, can be determined as follows:

$$\widehat{t}(m_i, k) = b_i \cdot T_{bus} + (s_i - 1) \cdot \Delta + (k - 1) \cdot r_i \cdot T_{bus}. \tag{2.39}$$

Thus, the first message instance starts at $\widetilde{t}(m_i, 1) = b_i \cdot T_{bus} + (s_i - 1) \cdot \Delta$ where (i) the term $b_i \cdot T_{bus}$ evaluates to the time elapsed before the start of the base cycle and (ii) the term $(s_i - 1) \cdot \Delta$ evaluates to the time elapsed from the start of the base cycle to the start of the slot in which the first instance of the message instance is mapped. Now after the first instance, the message can be sent every r_i cycles or $r_i \cdot T_{bus}$ time as given by the third term in Eq. (2.39). In the static

4Note that the cycle counter in FlexRay starts counting from 0.

segment, the message is sent entirely within a static slot. Thus, it can be guaranteed that the k-th instance of a message m_i will reach the destination ECU before a time $\widetilde{t}(m_i, k)$ that marks the end of the slot. Thus, $\widetilde{t}(m_i, k)$ is given by:

$$\widetilde{t}(m_i, k) = \widehat{t}(m_i, k) + \Delta = b_i \cdot T_{bus} + (k-1) \cdot r_i \cdot T_{bus} + s_i \cdot \Delta. \tag{2.40}$$

For a message m_i mapped on the dynamic segment, we can determine the earliest start time $\widehat{t}(m_i, k)$ of the k-th instance of the message as follows:

$$\widehat{t}(m_i, k) = b_i \cdot T_{bus} + N_s \cdot \Delta + (s_i - N_s - 1) \cdot \delta + (k-1) \cdot r_i \cdot T_{bus}. \tag{2.41}$$

This corresponds to the case where no data is sent in the preceding slots. Thus, $s_i - N_s - 1$ minislots are wasted before the slot counter shows s_i. On the other hand, we can determine the latest finish time $\widetilde{t}(m_i, k)$ of the k-th message instance as follows:

$$\widetilde{t}(m_i, k) = b_i \cdot T_{bus} + N_s \cdot \Delta + (\sum_{j=N_s+1}^{s_i} c_j) \cdot \delta + (k-1) \cdot r_i \cdot T_{bus}, \tag{2.42}$$

where c_j is the number of minislots required to transmit the message mapped on the slot s_j in the cycle number $b_i + (k-1) \cdot r_i$. Here, we consider the worst case where all messages are sent on the preceding slots in the cycle where the k-th message instance is assigned the slot. For example, in Figure 2.5, m_6 starts the earliest at the second minislot in cycle 4, while it finishes after the sixth minislot in the worst-case in cycle 0. Note that for a message mapped later on the dynamic segment would have a significant jitter. This must be taken into account while designing the controller or for analyzing the control performance.

2.3 Charge Transfer Circuits for Active Cell Balancing

Existing cell balancing approaches could be classified as either passive or active [163]. In comparison to passive techniques, where the excess charge in all cells is dissipated as heat across a high power resistor, active cell balancing approaches are energy-efficient, since they redistribute the charge among the cells instead of wasting them as heat. This transfer of charge between cells is facilitated using temporary energy storage elements such as *inductors, capacitors or transformers* coupled with a *power metal-oxide-semiconductor field-effect transistor* (MOSFET) *switching network.* Several circuit architectures with varying charge transfer features such as balancing between adjacent cells [164], direct charge transfer between non-adjacent cells [85], and charge transfers between groups of cells [165] have been proposed in the literature. In our work, we study two specific circuit architectures. In this section, we briefly discuss the working principle of the active cell balancing architectures under study. We further derive the necessary closed-form equations for charge transfer using these circuits. This will be later used in Chapter 6 to formulate the optimization problem for energy- and time-efficient active cell balancing.

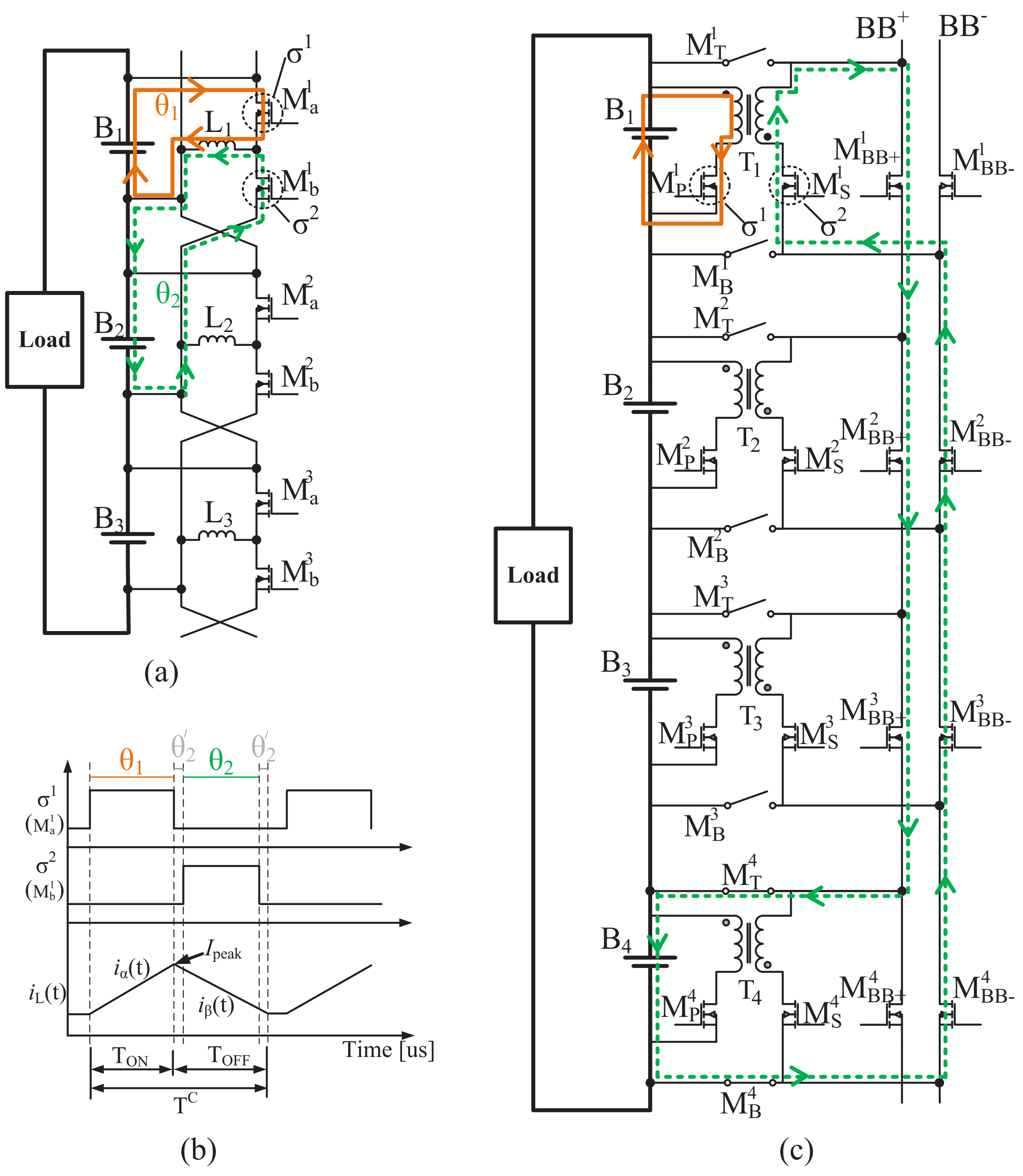


Figure 2.6: State-of-the-art charge transfer circuit architectures for active cell balancing. (a) This represents a neighbor-only cell balancing architecture [164]. A charge transfer cycle between the source cell $\mathcal{B}_1$ and the destination cell $\mathcal{B}_2$ is shown. The brown solid line depicts the charging phase when the inductor is charged by the cell $\mathcal{B}_1$. The green dashed line shows the discharging phase when the inductor discharges to the destination cell $\mathcal{B}_2$ (b) This shows the PWM control signals and the variation of the balancing current through the inductor during a charge transfer cycle. (c) This represents a non-neighbor active cell balancing architecture [85]. A cycle of charge transfer from $\mathcal{B}_1$ to $\mathcal{B}_4$ is shown. The brown solid line and the green dashed line depict the charging and the discharging phase respectively.

2.3.1 Charge Transfer Circuit Architectures

Neighbor-only balancing architecture: Figure 2.6(a) shows the state-of-the-art active cell balancing architecture that can transfer charge only between neighboring cells of the battery pack [164]. This circuit architecture works on the principle of buck-boost chopper [166]. The circuit architecture is modular in the sense that for each cell, there is an inductor and two MOSFET switches. For example, cell $\mathcal{B}_1$ is connected to L_1, M_a^1 and M_b^1. In the example in Figure 2.6(a), we show charge transfer from cell $\mathcal{B}_1$ to cell $\mathcal{B}_2$. A single charge transfer cycle (T^C) primarily consists of two phases, (i) the charging phase θ_1 (T_{ON}) and (ii) the discharging phase θ_2 (T_{OFF}), controlled by complimentary PWM control signals σ^1 and σ^2 respectively, as shown in Figure 2.6(b). One charge transfer cycle between the source cell $\mathcal{B}_1$ and the destination cell $\mathcal{B}_2$ can be described as follows:

- During θ_1, the switch M_a^1 is actuated with σ^1 and the excess charge in cell $\mathcal{B}_1$ is stored in the inductor L_1, as shown in Figure 2.6(a). The balancing current through the inductor starts to increase linearly and the switch M_a^1 is opened when the current reaches a peak value I_{peak}, as shown in Figure 2.6(b).

- During phase θ_2, the stored energy in the inductor is discharged to the destination cell $\mathcal{B}_2$ by actuating the MOSFET M_b^2 with the complimentary PWM control signal σ^2, as shown in Figure 2.6(a).

- Short free-wheeling phases θ_2' between the two control signals during which the inductor current flows through the body-diodes of the MOSFET switches are required to prevent short-circuit conditions.

Non-neighbor balancing architecture: The neighbor-only balancing architecture is energy-inefficient, when it is required to shuttle charge through multiple cells for charge equalization. By contrast, directly transferring charge between non-adjacent cells in a series-connected battery pack will provide a higher energy efficiency. Figure 2.6(c) shows such a balancing architecture [85] that can transfer charge directly between non-adjacent cells in a series-connected battery pack. In this balancing architecture, transformers are used as energy storage elements due to its inherent isolation properties that enables easy isolation between the balancing and the load circuits. The direct non-neighbor balancing is facilitated by having a charge transfer bus to which the cells are connected for exchanging charge between them and this bus is isolated at multiple stages to allow concurrent balancing in a series-connected battery pack. Here, each balancing unit corresponding to one cell in the series-connection comprises one transformer and eight MOSFET switches. In Figure 2.6(c), corresponding to $\mathcal{B}_1$, there is a transformer T_1 and the switches M_P^1, M_S^1, M_B^1, M_T^1, M_{BB+}^1 and M_{BB-}^1. Note that M_B^1 and M_T^1 are composed of two switches each. The primary winding of the transformer is connected to the cell $\mathcal{B}_1$ via the switch M_P^1 while the secondary winding is connected to the charge transfer bus via the switch M_S^1. The two switches in M_T^1 and M_B^1 respectively are connected in a way such that their internal body diodes are blocking each other, thereby completely isolating the cell from the charge transfer bus. The switches M_{BB+}^1 and M_{BB-}^1 provide isolation between the balancing units, thus, concurrent charge transfers are possible using the charge transfer bus.

In the example in Figure 2.6(c), cell $\mathcal{B}_1$ directly transfers charge to cell $\mathcal{B}_4$ through the charge transfer bus (BB^+ and BB^-). For this charge transfer, the bus is connected between the cells $\mathcal{B}_1$ and $\mathcal{B}_4$, i.e., the switches M^1_{BB+}, M^2_{BB+}, M^3_{BB+}, M^2_{BB-}, M^3_{BB-} and M^4_{BB-} are actuated. Now, during the charging phase, M^1_P is actuated using the control signal σ_1. Charge from cell $\mathcal{B}_1$ is stored in the primary winding of the transformer T_1 where the primary winding current increases linearly. After a predefined peak current value (I_{peak}) is reached for a time period of T_{ON}, M^1_P is turned OFF. Due to the property of the transformer, where the current cannot be interrupted instantaneously, the polarity of the voltages across the primary and secondary winding reverses. This polarity reversal causes the internal body-diode of the secondary winding MOSFET M^1_S to be forward-biased. As a result, the primary-secondary energy transfer takes place and the charge stored in the transformer is discharged to cell $\mathcal{B}_3$ through the secondary winding. This phase is marked as the short freewheeling phase. To minimize the loss involved in balancing over the diode, the discharging phase is initiated by actuating the secondary winding MOSFET M^1_S by the control signal σ_2. Now the transformer discharges and the stored energy is transferred into the cell $\mathcal{B}_4$ through the low-resistance MOSFET channel, as shown in Figure 2.6(c). Finally, the last remaining amount of charge stored in the transformer is discharged over the body-diode of M^1_S in order to prevent the transformer being charged from cell $\mathcal{B}_3$. This completes a charge transfer cycle using non-neighbor cell balancing architecture.

2.3.2 Modeling of Charge Transfer

The working principle and the modeling approach for both inductor- and transformer-based active cell balancing architectures are similar. Therefore, the generalized modeling approach is provided in this section. Modeling the charge transfer process between cells has been extensively studied in the literature so far [11, 165, 167, 168]. We follow a similar closed-form modeling approach considering the parasitic resistances of the circuit components in the balancing architecture.

Charging phase: Applying Kirchoff's voltage law [169] to the charging phase, we get:

$$V_\alpha = R_\alpha \cdot i_\alpha(t) + L \cdot \frac{d}{dt} i_\alpha(t), \tag{2.43}$$

where (i) V_α is the voltage of the source cell $\mathcal{B}_\alpha$; (ii) R_α is the sum of the parasitic resistances of the circuit components in the current flow path, thus, the term $R_\alpha \cdot i_\alpha(t)$ gives the sum of the voltage drops across the resistances; and (iii) L is the inductance, and therefore, the term $L \cdot \frac{d}{dt} i_\alpha(t)$ gives the voltage drop across the inductor (or transformer). R_α for the architectures shown in Fig. 2.6a and Fig. 2.6c is given by:

$$R_\alpha = R_{\mathcal{B}_\alpha} + R_L + R_M, \tag{2.44}$$

where $R_{\mathcal{B}_\alpha}$ is the resistance of the Li-ion cell, R_L and R_M are the parasitic resistances of the inductor and the MOSFET switch, respectively.

We assume that no charge was stored in the inductor (or transformer) at the beginning of the charge transfer cycle and the initial value of the current is $i_\alpha(0) = 0$. Solving Eq. (2.43) with this initial value gives a time-domain representation of the balancing current as follows:

$$i_\alpha(t) = \frac{V_\alpha}{R_\alpha}\left(1 - e^{-\frac{R_\alpha}{L}t}\right). \tag{2.45}$$

Using Eq. (2.45), we can determine the time T_{ON} of the control signal σ^1 such that the balancing current i_α reaches the saturation limit of the energy storage element I_{peak} as follows:

$$\begin{aligned}
& I_{peak} = \frac{V_\alpha}{R_\alpha}\left(1 - e^{-\frac{R_\alpha}{L}T_{ON}}\right) \\
\implies\quad & I_{peak} \cdot R_\alpha - V_\alpha = -V_\alpha e^{-\frac{R_\alpha}{L}T_{ON}} \\
\implies\quad & e^{-\frac{R_\alpha}{L}T_{ON}} = \frac{V_\alpha - I_{peak} \cdot R_\alpha}{V_\alpha} \\
\implies\quad & T_{ON} = \frac{L}{R_\alpha} \cdot \ln\left[\frac{V_\alpha}{V_\alpha - I_{peak} \cdot R_\alpha}\right].
\end{aligned} \tag{2.46}$$

By integrating $i_\alpha(t)$, given by Eq. (2.45), for the time period $0 \le t \le T_{ON}$, we obtain the total charge (Q_{tx}) transferred by the source cell as follows:

$$\begin{aligned}
Q_{tx} &= \int_0^{T_{ON}} i_\alpha(t) \cdot dt \\
&= \int_0^{T_{ON}} \frac{V_\alpha}{R_\alpha}\left(1 - e^{-\frac{R_\alpha}{L}t}\right) \\
&= \frac{V_\alpha}{R_\alpha} \cdot T_{ON} - \frac{L \cdot V_\alpha}{R_\alpha^2}\left[1 - e^{-\frac{R_\alpha}{L}T_{ON}}\right].
\end{aligned} \tag{2.47}$$

Substituting T_{ON} from Eq. (2.46), we get:

$$Q_{tx} = \frac{L \cdot V_\alpha}{R_\alpha^2} \ln\left[\frac{V_\alpha}{V_\alpha - I_{peak} \cdot R_\alpha}\right] - \frac{L \cdot I_{peak}}{R_\alpha}. \tag{2.48}$$

Discharging phase: Similar to the charging phase, when we apply Kirchhoff's voltage law [169] during the discharging phase, we get:

$$L \cdot \frac{d}{dt} i_\beta(t) + R_\beta \cdot i_\beta(t) + V_\beta = 0, \tag{2.49}$$

where V_β is the voltage of the destination cell, $i_\beta(t)$ is the balancing current during the discharging phase, and R_β is the sum of the parasitic resistances in the discharge circuit. For neighbor-only balancing architecture R_β is equal to R_α and is given by Eq. (2.44). However, for the non-neighbor balancing architecture, R_β is given by:

$$R_\beta = R_{\mathcal{B}_\beta} + R_L + (7 + 2 \cdot \check{d}) \cdot R_M, \tag{2.50}$$

where $R_{\mathcal{B}_\beta}$ is the resistance of the destination cell and $\check{d}$ is the number of cells in between the source and destination cells. Here, we assume that all the MOSFET switches are identical.

In the example in Figure 2.6(c), there are 11 MOSFET switches in the discharging circuit as follows: M^1_{BB+}, M^2_{BB+}, M^3_{BB+}, M^2_{BB-}, M^3_{BB-}, M^4_{BB-}, M^1_S, two MOSFETs each in M^4_T and M^4_B respectively.

The expression for i_β during the discharging phase can be obtained by solving the differential equation in Eq. (2.49) with the initial condition $i_\beta(0) = I_{peak}$, which is given as follows:

$$i_\beta(t) = I_{peak} \cdot e^{\frac{-R_\beta}{L}t} - \frac{V_\beta}{R_\beta}\left(1 - e^{\frac{-R_\beta}{L}t}\right). \tag{2.51}$$

The time T_{OFF}, required by the energy storage elements to fully discharge its stored energy to the destination cell, is calculated by substituting $i_\beta(T_{OFF}) = 0$ in Eq. (2.51). Thus, T_{OFF} is derived as follows:

$$\begin{aligned}
& 0 = I_{peak} \cdot e^{\frac{-R_\beta}{L}T_{OFF}} - \frac{V_\beta}{R_\beta}\left(1 - e^{\frac{-R_\beta}{L}T_{OFF}}\right) \\
\Longrightarrow \quad & I_{peak} \cdot e^{\frac{-R_\beta}{L}T_{OFF}} = \frac{V_\beta}{R_\beta}\left(1 - e^{\frac{-R_\beta}{L}T_{OFF}}\right) \\
\Longrightarrow \quad & \left(I_{peak} + \frac{V_\beta}{R_\beta}\right) \cdot e^{\frac{-R_\beta}{L}T_{OFF}} = \frac{V_\beta}{R_\beta} \\
\Longrightarrow \quad & T_{OFF} = \frac{L}{R_\beta} \cdot \ln\left[\frac{V_\beta + I_{peak} \cdot R_\beta}{V_\beta}\right].
\end{aligned} \tag{2.52}$$

The charge received by the destination cell (Q_{rx}) is obtained by integrating i_β, given in Eq. (2.51), for the time period $0 \le t \le T_{OFF}$, which is given by:

$$\begin{aligned}
Q_{rx} &= \int_0^{T_{OFF}} i_\beta(t) \cdot dt \\
&= \int_0^{T_{OFF}} \left(I_{peak} \cdot e^{\frac{-R_\beta}{L}t} - \frac{V_\beta}{R_\beta}\left(1 - e^{\frac{-R_\beta}{L}t}\right)\right) \cdot dt \\
&= \frac{L \cdot V_\beta}{R^2_\beta} \ln\left[\frac{V_\beta}{V_\beta + I_{peak} \cdot R_\beta}\right] + \frac{L \cdot I_{peak}}{R_\beta}.
\end{aligned} \tag{2.53}$$

Note that the change in voltage in the source and destination cells during a charge transfer cycle is less than $1\,\mathrm{nV}$. For a nominal cell voltage of 3.6V, this will have a negligible impact on the values of Qtx and Qrx in Eq. (2.48) and Eq. (2.53) respectively.

2.4 Optimization and Verification

The main goal of this thesis is to devise techniques that enable optimal design of CPSs. In the process, we have proposed hybrid optimization techniques customized for the design problems under study. These proposed design approaches combine different well-established optimization and verification models by suitably applying them to specific sub-problems. In this section, we will provide a brief background on these existing models that will help the readers to better understand the intuition behind applying these modeling approaches to our problem settings.

2.4.1 Linear Programming

We can formulate a *linear programming* (LP) model for a mathematical optimization or a constraint satisfaction problem. An LP model consists of (i) decision variables $\{var_1, var_2, ..., var_{N_v}\}$, (ii) an objective function Obj and (iii) constraints $con_1, con_2, \cdots, con_{N_c}$. Note that, in an LP problem, the constraints and the objective function are linear in variables. An LP problem in its canonical form can be written as follows:

$$\textbf{Objective:} \quad \text{Maximize} \quad \sum_{j=1}^{N_v} c_j \cdot var_j$$

$$\textbf{Constraints:} \quad \sum_{j=1}^{N_v} a_{i,j} \cdot var_j \leq b_i, \qquad (i = 1, 2, 3, \cdots, N_c)$$

$$\textbf{Variables:} \quad var_j \in \mathbb{R} \qquad (j = 1, 2, 3, \cdots, N_v)$$

In a *mixed-integer linear programming* (MILP) problem, a subset of variables can only assume integer values. The variable declaration in an MILP model can be written as:

$$\begin{aligned} \textbf{Variables:} \quad & var_j \in \mathbb{R}, && (j = 1, 2, 3, \cdots, N_{vr}) \\ & var_j \in \mathbb{Z}, && (j = N_{vr}+1, N_{vr}+2, N_{vr}+3, \cdots, N_{vr}+N_{vi}). \end{aligned}$$

Moreover, we can also formulate a linear model with integer variables only. Such a formulation is termed as *integer linear programming* (ILP). The variable declaration in an ILP model can be written as:

$$\textbf{Variables:} \quad var_j \in \mathbb{Z} \qquad (j = 1, 2, 3, \cdots, N_v)$$

For more details on linear optimization, please refer to [170].

Example: Consider a linear optimization problem as shown below:

Minimize $\quad -2 \cdot var_1 - 3 \cdot var_2$

subject to:

$$\begin{aligned} & 2 \cdot var_1 + var_2 + var_3 = 4 \\ & 3 \cdot var_1 - 9 \cdot var_2 + var_4 \geq 4 \\ & 3 \cdot var_1 + 2 \cdot var_3 + var_4 \leq 35 \\ & var_1, var_2, var_3, var_4 \geq 0 \end{aligned}$$

$var_1, var_2 \in \mathbb{R}$ and $var_3, var_4 \in \mathbb{Z}$

The above example comprises four variables out of which var_3 and var_4 are integer variables while var_1 and var_2 can take real values. Note that the objective is to minimize a linear function of decision variables, however, it can be translated into the canonical form as follows:

$$\text{Maximize} \quad 2 \cdot var_1 + 3 \cdot var_2,$$

where we just reverse the signs of the coefficients. Moreover, in case we want to formulate a constraint satisfaction problem as an LP, ILP or MILP model, we can write the objective function as a constant. With a constant objective, a value assignment to the decision variables that satisfies all constraints will be returned as the solution. Note that the constraints in the example are either a linear equality (=) or a linear inequality ($\geq$ or $\leq$). In the canonical form, we have only "less than or equal to ($\leq$)" constraint. The "greater than or equal to ($\geq$)" constraint in the example can be transformed into the canonical form as follows:

$$\begin{aligned} & 3 \cdot var_1 - 9 \cdot var_2 + var_4 \geq 4 \\ \implies \quad & -3 \cdot var_1 + 9 \cdot var_2 - var_4 \leq -4. \end{aligned}$$

On the other hand, the equality constraint in the example can be translated to the canonical form as follows:

$$2 \cdot var_1 + var_2 + var_3 = 4$$

$$\begin{aligned} \implies \quad & 2 \cdot var_1 + var_2 + var_3 \leq 4 \quad && \text{and} \quad && 2 \cdot var_1 + var_2 + var_3 \geq 4 \\ \implies \quad & 2 \cdot var_1 + var_2 + var_3 \leq 4 \quad && \text{and} \quad && -2 \cdot var_1 - var_2 - var_3 \leq -4. \end{aligned}$$

Here, we need two "less than or equal to" constraints to represent one "equality" constraint. The above example can be, therefore, formulated as an MILP model.

Solvers: There are several commercial and non-commercial solvers available for solving LP, MILP and ILP problems. For our experiments, we have used *CPLEX* [171] and *Gurobi* [172]. These solvers accept problem models in specific forms and return solutions based on the model specifications. The underlying algorithmic implementation of these solvers are beyond the scope of this thesis.

2.4.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is a popular method to solve non-convex non-linear optimization problems. It is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995 [173], inspired by social behavior of bird flocking or fish schooling. It is an example of how a computational technique can be developed based on the bahavior of a biological system.

PSO is a DSE technique where a group of particles, i.e., candidate solutions, are initialized randomly in the design space. These particles then search for the optima by moving in the design space based on certain update equations. Each particle has a position, a velocity and a fitness value. Position determines the current values of the decision variables. Velocity determines how the position of the particle would change next. Fitness value is the value of the objective corresponding to the current position.

In every iteration, each particle's position is updated according to the velocity which is calculated primarily based on two values. The first one is the best solution (in terms of the fitness value) the particle has achieved so far. This value is called the *personal best* denoted by $pbest$. The second one that is tracked by the optimizer, is the best solution obtained so far by any particle in the population. This value is the *global best* which is denoted by $gbest$.

While implementing the PSO algorithm, we need to define a set of rules to compare different solutions for finding the $pbest$ and the $gbest$. In our implementation, we let feasibility dominate performance. The specified rules can be written as follows:

- A value assignment respecting all constraints is better than another assignment violating one or more constraints. That is, the objective value has no influence.
- If two solutions respect all constraints, then the solution with a better fitness value (i.e., the optimization objective) is considered better.
- If two value assignments do not satisfy the constraints (i.e., both of them violate at least one constraint), the value assignment with a better fitness value is considered better.

Let us denote a particle as par_i and its current position and velocity as pos_i and vel_i. A particle updates its position and velocity according to the following equations:

$$vel_i = \alpha_0 \cdot vel_i + \alpha_1 \cdot rnd(0,1) \cdot (pbest_i - pos_i) + \alpha_2 \cdot rnd(0,1) \cdot (gbest - pos_i), \quad (2.54)$$

$$pos_i = pos_i + vel_i. \quad (2.55)$$

Here, (i) $rnd(0,1)$ is a random number with uniform distribution from the open interval $(0,1)$, (ii) $pbest_i$ is the personal best of the particle par_i, (iii) α_0 is the weight inertia and α_1 and α_2 are cognitive and social scaling parameters respectively. Widely used values for these parameters are: $\alpha_0 = 0.4$ and $\alpha_1 = \alpha_2 = 2$ [174]. The algorithm is terminated once all particles have converged or the maximum number of iterations has been reached. The computational complexity of PSO is clearly polynomial.

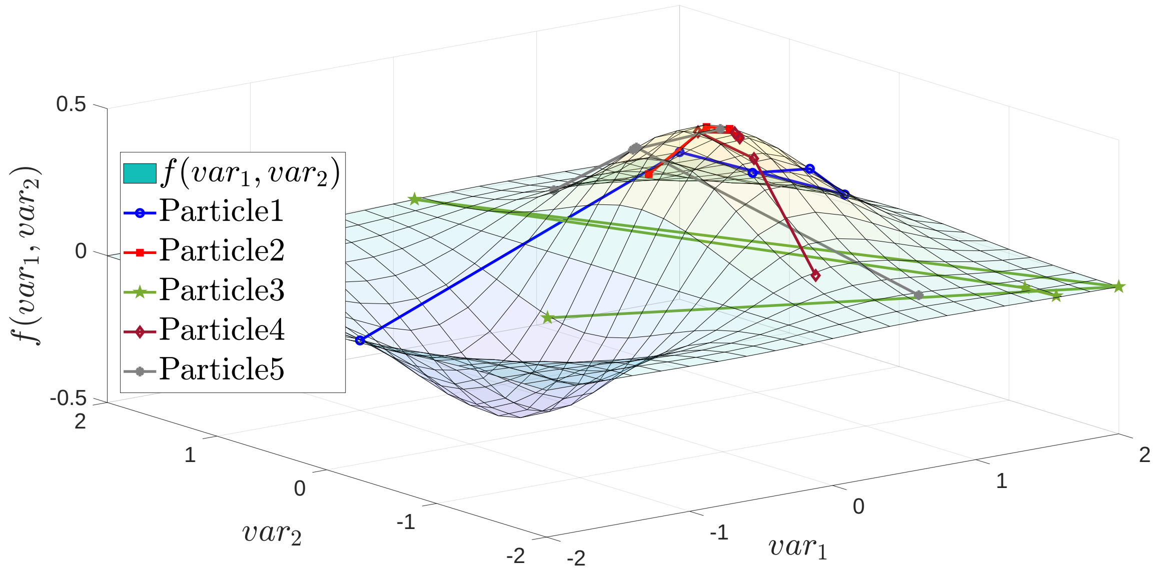


Figure 2.7: Nonlinear optimization using particle swarm optimization (PSO). Five particles are used for the optimization and their search trajectories are shown. ‘Particle2’, ‘Particle4’ and ‘Particle5’ are very close to the maxima.

Example: Let us consider a nonlinear objective function as follows:

$$f(var_1, var_2) = var_1 \cdot e^{-var_1^2 - var_2^2},$$

where, $-2 \leq var_1 \leq 2$ and $-2 \leq var_2 \leq 2$. We apply PSO to maximize the value $f(var_1, var_2)$ within the bounds of the variables var_1 and var_2. We use five particles and five iterations and their search trajectories are as shown in Figure 2.7. It can be observed that four particles are approaching the maxima in only five iterations. The best result is obtained by 'Particle5' which is $f(var_1, var_2) = 0.4277$ while the optimal value of $f(var_1, var_2)$ is 0.4289.

2.4.3 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) is a powerful modeling approach for constraint satisfaction and verification problems. While a *boolean satisfiability* (SAT) problem is modeled as a propositional logic formula, an SMT model can express a first-order logic formula [175].

A SAT problem comprises only boolean variables and is typically expressed as a boolean formula containing variables in conjunctive normal form (CNF). A solution to the SAT problem is a set of value assignments to the corresponding set of variables such that the boolean formula evaluates to true. Let us consider an example as follows:

$$(a' \vee b \vee c) \wedge (a \vee c \vee d) \wedge (a \vee c \vee d') \wedge (a \vee c' \vee d) \wedge (a \vee c' \vee d') \wedge (b' \vee c' \vee d) \wedge (a' \vee b \vee c') \wedge (a' \vee b' \vee c)$$

A solution to this SAT problem is as follows:

$$\{a \Leftrightarrow \mathrm{T}, b \Leftrightarrow \mathrm{T}, c \Leftrightarrow \mathrm{T}, d \Leftrightarrow \mathrm{T}\}^5$$

SMT problems can be constructed with non-boolean variables as well. In a SAT problem, if we replace boolean variables with predicates over non-boolean variables then it becomes an SMT problem. A predicate evaluates to a binary value and can be an expression with non-boolean variables. Examples of such predicates include:

- Arithmetic expressions, i.e., linear and non-linear equalities and inequalities with integer and real variables, such as

  (i) $var_1 - 3 \cdot var_2 + 4 \cdot var_3 \geq 10,$ (ii) $2 \cdot var_1 \cdot var_2 + 4 \cdot var_1 \cdot var_2^2 - 5 \cdot var_3 = 5.$

  For such a predicate, the solution would comprise the value assignment to the variables such that the corresponding arithmetic expression evaluates to true. For example, the following value assignment will make the above arithmetic predicates to be true.

  $$var_1 = 1, \quad var_2 = -3, \quad var_3 = 5.$$

- Logical, arithmetic and structural operations on bitvectors [176], e.g.,

  (i) $var_1 \vee \neg var_2 = var_3,$ (ii) $var_1 \cdot var_2 = var_3,$ (iii) $\mathrm{SHL}(var_1, var_4) = var_3.$

[5] T refers to a "true" value assignment

Here, var_1, var_2 and var_3 are bit vectors while var_4 is a natural number. The first predicate states that bitwise-or between var_1 and var_2 must be equal to var_3. If the number of bits in a bitvector is equal to k, then the second predicate states that $\mathcal{I}(var_1) \cdot \mathcal{I}(var_2) \mod 2^k - 1 = \mathcal{I}(var_3)$, where $\mathcal{I}(\cdot)$ is the integer equivalent of a bitvector. The third predicate implies that when the bits of the bitvector var_1 are shifted var_4 positions to the left then we must get var_3. Predicates on bitvectors are usually required to model software verification problems.

- Predicates over arrays [177] involving read and write operations such as

  $$[\text{read}(arr, ind) = elem_1] \wedge [\text{write}(arr, ind, elem_2) = arr] \wedge [\neg(elem_1 = elem_2)] .$$

  Here, arr denotes an array, ind is an array index and $elem_1$ and $elem_2$ are values. In the above example, there are three predicates. The first one says that when the array arr is read at the index ind, we should get the value $elem_1$. The second predicate says that we write the value $elem_2$ at the index ind of the array arr. Thus, these predicates can both be true only when $elem_1$ and $elem_2$ are equal. However, this is not true according to the third predicate. Thus, there is no solution to the above example as the constraints are unsatisfiable.

- Expressions involving uninterpreted functions [178] such as

  $$var_1 \cdot [f(var_2) + f(var_3)] = var_2 \ \wedge \ var_2 \cdot [f(var_1) + f(var_3)] \neq var_2 \ \wedge \ var_1 = var_2.$$

  Here, $f(\cdot)$ is an uninterpreted function as the input-output relation is not defined for the function. However, it is possible to determine that the above formula is unsatisfiable. This is because the first two predicates can be true only if $var_1 \neq var_2$, however, this violates the third constraint.

Thus, it is possible to formulate complex constraint satisfaction problems using SMT models compared to SAT and LP because different constraints using different types of variables are allowed in SMTmodels.

Solvers: Earlier SMT techniques, like in MathSAT [179] and iSAT [180], are based solely on Davis-Putnam-Loveland-Logemann (DPLL) algorithm that is traditionally used for solving SAT problems. This is called the eager approach [175] where an SMT formula is first translated into an equivalent SAT formula and then DPLL algorithm [181] is used to solve the problem. For example, a 32 bit integer value is represented by 32 binary variables and higher level integer addition is replaced by lower level boolean operation. However, this approach is inefficient and often a solver runs out of memory or time because of the complexity of the problem. Thus, modern SMT solvers, like Z3 [182] and Yices [183], employ a combination of SAT solving and theory solving (T-solver) techniques, which is known as the lazy approach [175]. In this technique, a proposition can contain predicates involving real, integer, arrays, bitvectors or uninterrupted functions. Here, SAT solver evaluates boolean relations while different theory solvers (T-solvers) are used to evaluate other relations. For example, the theory of real numbers is used to evaluate predicates with arithmetic expressions comprising real numbers while theory of arrays is used to solve predicates over arrays. relations. T-solvers interact closely in the lazy approach that makes it very efficient.

2.4.4 Timed Automata

Real-time systems can be formally modeled using timed automata (TAs). A timed automaton (TA) is a finite state machine with a finite set of real-valued clocks that progress synchronously. A TA can be represented as a tuple $(LOC, loc_0, ClK, ACT, EDG, INV)$ [184] where LOC is a set of locations, loc_0 is the initial location, CLK is a set of clocks, ACT is a set of actions, $EDG \subseteq LOC \times ACT \times grd(CLK) \times 2^{CLK} \times LOC$ is a set of edges between locations with an action, a guard and a set of clocks to be reset respectively, and $INV : LOC \longrightarrow GRD(CLK)$ assigns invariants to locations. A clock valuation is a function $cval : CLK \longrightarrow \mathbb{R}^{CLK}_{\geq 0}$ from the set of clocks to non-negative reals. Here, $grd(CLK)$ is a set of conjunctions over simple conditions of the form $clk_i \bowtie val$ or $clk_i - clk_j \bowtie val$ where $clk_i, clk_j \in CLK$, $val \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Considering invariants as sets of clock valuations, $cval \in INV(loc_i)$ means that $cval$ satisfies $INV(loc_i)$, i.e., the invariant corresponding to the location loc_i.

The operation semantics of a TA is defined as a labeled transition system $\langle ST, st_0, \rightarrow \rangle$ [184], where $ST \subseteq LOC \times \mathbb{R}^{CLK}$ is the set of states, st_0 is the initial state, and $\rightarrow \subseteq ST \times (\mathbb{R}_{\geq 0} \cup ACT) \times ST$ is the transition relation such that:

- $(loc, cval) \xrightarrow{del} (loc, cval + del)$ if $\forall del' : 0 \leq del' \leq del \Rightarrow cval + del' \in INV(loc)$, and
- $(loc, cval) \xrightarrow{act} (loc', cval')$ if there exists $edg = \{loc, act, grd, rst, loc'\} \in EDG$ such that $cval \in grd$, $cval' = [RST \mapsto 0]cval$, and $cval' \in INV(loc')$,

where for $del \in \mathbb{R}_{\geq 0}$, $cval + del$ maps each clock $clk \in CLK$ to the values $cval(clk) + del$, and $[RST \mapsto 0]cval$ denotes the clock valuation which maps each clock in RST to 0 and agrees with $cval$ over $CLK \backslash RST$.

A number of TAs often compose a network over a common set of clocks and actions, consisting of N_{TA} TAs, where $TA_i = (LOC_i, loc_{i,0}, CLK_i, ACT_i, EDG_i, INV_i)$, $1 \leq i \leq N_{TA}$ [184].

Example: We consider a real-time system comprising several software tasks scheduled according to a fixed-priority preemptive scheduling policy on a processor. We can model the system as a network of TAs. Each task can be represented as a TA as shown in Figure 2.8(a). We can also model the scheduler as a TA as shown in Figure 2.8(b). The task automaton and the scheduler automaton are adapted from [185].

In Figure 2.8, the solid circles are locations, where two concentric circles denote an initial location and a location marked with 'C' is a committed location. A state becomes committed if it comprises a location that is committed. Time cannot elapse in a committed state, and therefore, an outgoing edge of one of the committed locations must be taken in the next transition. The black arrows connecting locations are the edges. Texts written in green along the edges are guards while texts in brown corresponding to locations are invariants. Two TAs communicate via synchronization channels (written in gray in the figure). Texts written in blue are the actions.

We consider that each task has a dispatch offset, a period, a worst-case execution time (WCET) and a best-case execution time (BCET). As shown in Figure 2.8(a), a task automaton starts in the location 'Initial' defined by the invariant that the time must be less than or equal to the task's dispatch offset. Thus, the automaton makes a transition to a committed location

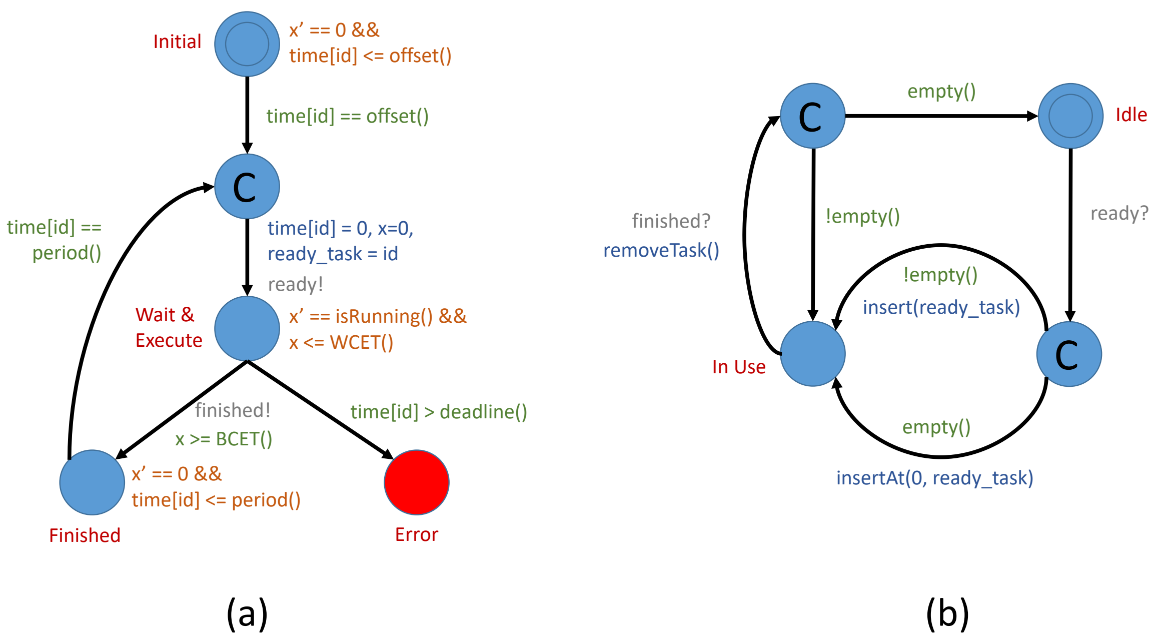


Figure 2.8: Modeling fixed-priority preemptive scheduling as a network of timed automata. (a) The timed automaton (TA) representing a task defined by its offset, period, worst-case execution time (WCET) and best-case execution time (BCET). (b) The timed automaton (TA) representing the fixed-priority preemptive scheduler.

only after a time equal to the dispatch offset has elapsed. The next transition has to be from the committed location where the clocks ('time[·]' and 'x') used by the task automaton are reset and it communicates to the scheduler automaton that it is ready to run using the synchronization channel 'ready'. In the location 'Wait & Execute', 'x' is used as a stopwatch which runs when the task is being run by the processor and it is paused when it is waiting. This location also has an invariant that the automaton can stay here only if the task has not been allocated to the processor for a time more than its WCET. If the task stays in this location for a time more than its deadline then the automaton takes the transition to the 'Error' state, which implies that the scheduling constraint is violated. If the time, the task ran on the processor is more than its BCET, it can move to the 'Finished' state and communicate this to the scheduler automaton using the synchronization channel 'finished'. The 'Finished' state has an invariant that the automaton must stay there till the task's next dispatch that is measured based on its period. For the next dispatch, the automaton again moves to the committed location.

As shown in Figure 2.8(b), the scheduler automaton starts in the 'Idle' state. Whenever a task is dispatched as communicated by the 'ready' synchronization channel, it moves to a committed location where it evaluates whether the ready queue is empty. If the buffer is empty, it adds the ready task at the beginning of the queue, otherwise, the ready task is added to the queue based on its priority. All dispatched tasks with a higher priority than the new task must be aheady in the queue while the lower-priority tasks must be behind the new task. After adding the task, the automaton moves to the location 'In Use' that implies that the processor is running a task that is at the beginning of the queue. When a task notifies the scheduler using the 'finished' synchronization channel that it has finished execution, the task is removed from the buffer and the automaton moves to a committed location. If the buffer is not empty, it again comes back to the 'In Use' state otherwise it makes a transition to the 'Idle' state.

Verification tools: There are a few tools available to verify properties of TAs such as UPPAAL [186] and Kronos [187]. In our work, we have used UPPAAL as the modeling and verification tool for TAs. In UPPAAL, properties can be specified using timed computation tree logic (TCTL). It can be used to verify safety, reachability, liveness properties as well as deadlocks. In the example in Figure 2.8, "A[] forall (i : task_id) not Task(i).Error" specifies that for all states neither of the task automata must be in the location 'Error'. We can use this specification to verify whether all tasks satisfy their respective deadlines.

2.4.5 Minimum Vertex Coloring

Let us consider an undirected graph composed of vertices and edges represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Here, two vertices $v_1, v_2 \in \mathcal{V}$ are connected by an edge $\{v_1, v_2\} \in \mathcal{E}$ only if a certain condition is satisfied. Note that an edge is denoted by an unordered pair of vertices.

With regard to an unidirected graph, *vertex coloring* is the problem of assigning colors to the vertices of the graph such that no two vertices connected by an edge have the same color (Chapter 6 in [188]). Given a graph, it is trivial to color the vertices based on the aforementioned rules. However, for real applications, typically, it is desirable to color the graph using the minimum number of colors. This is a challenging problem. Practical applications of *minimum vertex coloring* (MVC) include timetable formulation, mobile radio frequency assignment, map coloring, sudoku, and register allocation in compiler optimization.

Example: Suppose we need to formulate an exam timetable for a particular class. In total there are six subjects, i.e., english (E), science (S), mathematics (M), computer applications (C), histotry (H) and geography (G). Students can choose different subjects. Thus, no two exams can be scheduled simultaneously if there is at least one student who is registered for both exams. These scheduling constraints can be represented in a graph as shown in Figure 2.9(a). Here, each vertex represents a subject and if there is an edge between two vertices, it implies that the exams for the corresponding two subjects cannot take place at the same time. For example, mathematics and science cannot be scheduled together while mathematics and history can be simultaneously scheduled. Now, the task at hand is to determine the minimum number of time slots required to schedule all six exams.

This problem can be formulated as an MVC problem where a color represents a timeslot and we need to color the graph shown in Figure 2.9(a) such that no two vertices sharing an edge can have the same color. A possible solution to this problem is depicted in Figure 2.9(b) where we have used four colors which implies we need four timeslots to schedule the exams. According to the solution, mathematics and geography exams can share a timeslot, and science and history exam can be conducted simultaneously, however, english and computer application need a separate timeslot each.

MVC of chordal graphs: In graph theory, a chordal graph is a graph where every cycle with four or more vertices must have a chord (Chapter 6 in [188]). Consider four or more vertices $\mathcal{V}_{cy} = \{v_1, v_2, v_3, \cdots, v_n\}$ that form a cycle denoted by a set of edges $\mathcal{E}_{cy} =$

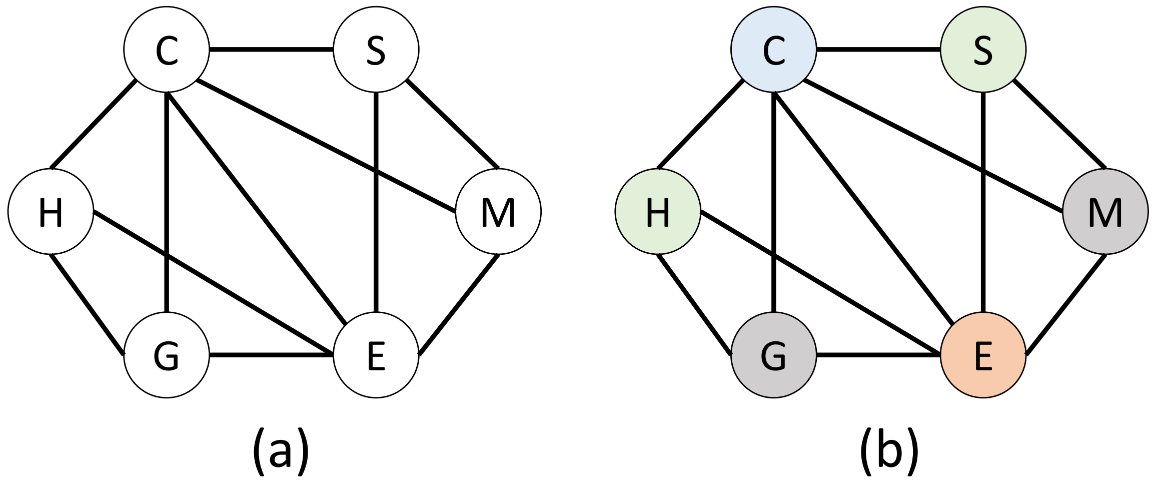


Figure 2.9: Application of minimum vertex coloring (MVC). (a) An undirected graph representing the scheduling constraints between exams on different subjects is shown. Each vertex represents a subject and two vertices connected by an edge implies that the exams for the corresponding two subjects cannot be given simultaneously. (b) A colored undirected graph is shown where each color represents a timeslot. The colored graph, therefore, illustrates the exam schedule. Two vertices with the same colored will be conducted at the same time. Four colors are used which implies that four timeslots are needed to schedule six exams.

$\{\{v_1, v_2\}, \{v_2, v_3\}, \cdots, \{v_i, v_{i+1}\}, \cdots, \{v_{k-1}, v_k\}, \{v_k, v_1\}\}$. Now, if this cycle is a part of a chordal graph $\mathcal{G}_{ch}$, then there exists at least an additional edge in the graph $\mathcal{G}_{ch}$ shared by two vertices in the set $\mathcal{V}_{cy}$, that is not a part of the cycle. This can be written as

$$\underset{v_i, v_j \in \mathcal{V}_{cy}}{\exists} (\{v_i, v_j\} \in \mathcal{E}_{ch}) \wedge (\{v_i, v_j\} \notin \mathcal{E}_{cy}). \tag{2.56}$$

The graph shown in Figure 2.9(a) is also a chordal graph. As can be seen in the figure, the cycle C-S-E-G-C has a chord E-C and the cycle H-C-E-G-H has two chords H-E and C-G.

MVC is generally an NP-hard problem. However, for chordal graphs, there exists a linear time algorithm for MVC [189]. Vertices of a chordal graph can be colored using the minimum number of colors in two steps. First, a *perfect elimination ordering of the vertices* (PEOV) of the graph is identified. Here, the ordered set $(v_1, v_2, \cdots, v_i, \cdots, v_n)$ is a PEOV if and only if each vertex v_i is simplicial in the corresponding subgraph $\mathcal{G}_{v_i}$ induced by the vertices $\{v_i, v_{i+1}, \cdots, v_n\}$. A vertex v_i is *simplicial* if v_i along with its adjacent vertices $Adj(v_i)$ in a graph form a clique. Here, $Adj(v_i)$ represents the set of vertices in $\mathcal{G}_{v_i}$ that v_i has edges with. Such an ordering can be obtained for chordal graphs in linear time using the *lexicographic breadth first search* (Lex-BFS) algorithm [190]. In the second step, the *greedy vertex coloring algorithm* [189] is applied backwards to the obtained PEOV. Here, considering that the colors are numbered as $\{1, 2, \cdots\}$, each vertex v_i is assigned a minimum numbered color that is not used by any of its adjacent vertices in $Adj(v_i)$.

Lex-BFS algorithm is given in Algorithm 1 that determines the PEOV in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The input of the algorithm is the set of vertices $\mathcal{V}$. The output is the PEOV $\mathcal{V}_{ord}$. We denote Ψ as an ordered set of sets of vertices such that the vertices in the first set will be at the end of the

Algorithm 1: Lexicographic breadth first search (Lex-BFS) algorithm.

```
Input  : $\mathcal{V}$
Output: $\mathcal{V}_{ord}$
1  $\Psi = \big(\mathcal{V}\big)$;
2  $n_v = \big|\mathcal{V}\big|$;
3  for $k \leftarrow 0$ to $n_v - 1$ do
4      $v_a$ = SelectVertex( $\Psi[1]$ );
5      $\mathcal{V}_{ord}\big[\, n_v - k \,\big] = v_a$;
6      $\Psi[1] = \Psi[1] \setminus v_a$;
7      $\Psi^*$ = NULL;
8      $j = 1$;
9      for $i \leftarrow 1$ to $\big|\Psi\big|$ do
10         $\mathcal{V}_c = \big\{ v_b \in \Psi[i] \,\big|\{v_a, v_b\} \in \mathcal{E} \big\}$;
11         $\mathcal{V}_{nc} = \Psi[i] \setminus \mathcal{V}_c$;
12         if $\mathcal{V}_c \neq \emptyset$ then
13             $\Psi^*[j] = \mathcal{V}_c$;
14             $j = j + 1$;
15         end
16         if $\mathcal{V}_{nc} \neq \emptyset$ then
17             $\Psi^*[j] = \mathcal{V}_{nc}$;
18             $j = j + 1$;
19         end
20     end
21     $\Psi = \Psi^*$;
22 end
```

PEOV. Moreover, Ψ will be refined in each iteration of the algorithm. In line 1, we initialize Ψ with only one set, i.e., the set of all vertices in the graph $\mathcal{V}$. In line 2, we determine the number of vertices n_v in the graph. Lex-BFS algorithm iteratively identifies a vertex that is simplicial with respect to the subgraph induced by it together with the already selected vertices (lines 3 to 22). Thus, it determines the PEOV in the reverse order. In each iteration, a vertex v_a is randomly selected from the first set of vertices in Ψ, i.e., $\Psi[1]$ (line 4). Then, v_a is placed in the ordered set $\mathcal{V}_{ord}$ before the vertices selected in the previous iterations (line 5) and is removed from the set $\Psi[1]$ (line 6). We denote Ψ^* as the new ordered set consisting of sets of remaining vertices and initialize it as an empty set (line 7). Ψ^* is populated iteratively based on the ordered sets in Ψ (lines 9 to 20). We denote j as the current index in Ψ^* where a new set can be added and initialize it as 1 (line 8). Now, each set $\Psi[i] \in \Psi$, is partitioned into two mutually exclusive subsets $\mathcal{V}_c$ and $\mathcal{V}_{nc}$, where $\mathcal{V}_c$ ($\mathcal{V}_{nc}$) consists of all vertices in $\Psi[i]$ that have (do not have) an edge with v_a (lines 10 and 11). If $\mathcal{V}_c$ is a non-empty set then we append it to Ψ^* and subsequently increment j by 1 (lines 12 to 15). Similarly, if $\mathcal{V}_{nc}$ is a non-empty set then we place it in Ψ^* in the j-th position and then increment j by 1 (lines 16 to 19). When all the sets in Ψ are traversed then we replace the current ordering of sets in Ψ with the new ordering in Ψ^* for the next iteration (line 21).

Let us consider the example in Figure 2.9(a). The Lex-BFS algorithm when applied to this example, Ψ and $\mathcal{V}_{ord}$ obtained at different stages of the algorithm are given as follows:

- Initialization: $\Psi = \big(\{E, S, M, C, H, G\}\big)$, $\mathcal{V}_{ord} = ()$.
- After iteration 1: $\Psi = \big(\{S, M, C, H, G\}\big)$, $\mathcal{V}_{ord} = (E)$.
- After iteration 2: $\Psi = \big(\{M, C\}; \{H, G\}\big)$, $\mathcal{V}_{ord} = (S, E)$.
- After iteration 3: $\Psi = \big(\{C\}; \{H, G\}\big)$, $\mathcal{V}_{ord} = (M, S, E)$.
- After iteration 4: $\Psi = \big(\{H, G\}\big)$, $\mathcal{V}_{ord} = (C, M, S, E)$.
- After iteration 5: $\Psi = \big(\{G\}\big)$, $\mathcal{V}_{ord} = (H, C, M, S, E)$.
- After iteration 6: $\Psi = \big(\big)$, $\mathcal{V}_{ord} = (G, H, C, M, S, E)$.

Now, we can apply the greedy vertex coloring algorithm to the obtained PEOV $\mathcal{V}_{ord}$ in the reverse order. Let us assume an ordered color sequence as (orange, green, gray, blue, $\cdots$). First, we color the vertex 'E' with orange. The vertex 'S' cannot have the same color as 'E' and, therefore, is colored green. The vertex 'M' cannot be colored the same as 'E' and 'S', and thus, a new color, i.e., gray, is selected. The vertex 'C' needs a new color, i.e., blue, as all the vertices traversed so far have edges with 'C'. The vertex 'H' cannot have the same color as 'E', however, it can be colored using green because it does not have an edge with 'S'. The vertex 'G' cannot be colored with orange, green and blue as it has edges with 'E', 'H' and 'C' respectively, and therefore, gray is used for 'G'. Overall four colors are required to color the graph as shown in Figure 2.9(b).

For a chordal graph, the minimum number of colors required to color all the vertices in the graph is given by the cardinality of the clique with the maximum number of vertices [191]. A clique consists of a subset of vertices in a graph where each vertex has an edge with every other vertices in it (Chapter 4 in [188]). Thus, the set of vertices in a clique induces a complete subgraph. In the context of vertex coloring, no two vertices in a clique can have the same color. Thus, the total number of colors required to color all the vertices in a clique is equal to the cardinality of the clique. Now, a maximal clique is a clique to which no further vertices can be added to form another clique [192]. Or, in other words, a maximal clique is not a proper subset of any other clique of the graph. Thus, in a chordal graph, the clique with the maximum cardinality, that determines the minimum number of colors required to color the vertices of the graph, is one of the maximal cliques. In the example in Figure 2.9a, there are two maximal cliques formed by (i) E, S, M, and C and (ii) E, C, H, and G respectively. Both maximal cliques comprise four vertices. Thus, the maximum number of color required for this example is equal to four which is also the number obtained by applying the Lex-BFS and greedy vertex coloring algorithm.

Interval graphs: Our work further studies *interval graphs* and exploits specific properties of such graphs. An interval graph is a graph where the vertices represent intervals on a real

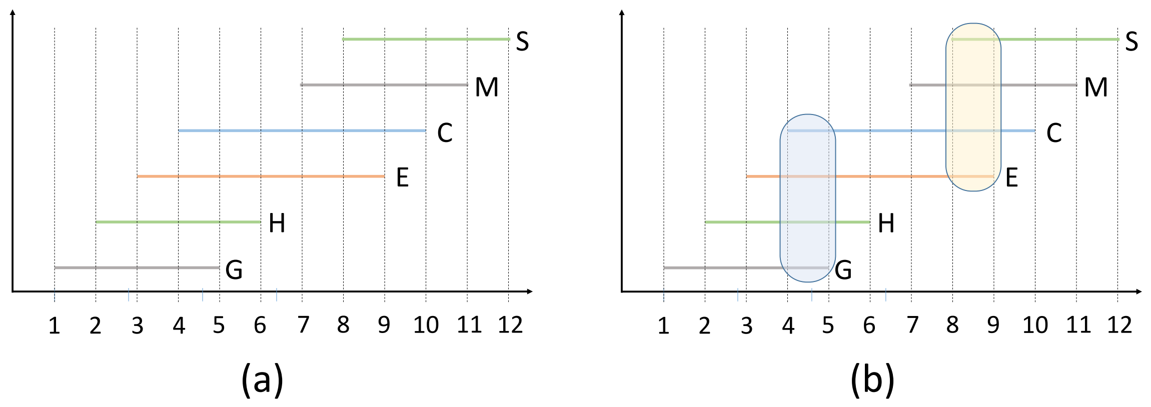


Figure 2.10: Minimum vertex coloring of interval graphs. (a) Six interval are shown that are colored such that no two overlapping intervals have the same color. (ii) Two maximal cliques are determined in the interval graph induced by the intervals. The first maximal clique is formed by G, H, E, and C as these intervals contain $]4, 5[$ where 4 is a left endpoint and 5 is a right endpoint. The second maximal clique comprises E, C, M, and S and these intervals cover $]8, 9[$ where 8 is a left endpoint and 9 is a right endpoint.

line, and when two intervals overlap, the corresponding two vertices will be connected by an edge [193]. Note that all interval graphs are chordal [194]. Thus, the number of colors required to color the vertices of an interval graph is given by the cardinality of the maximum clique. [192] provides a theorem to determine all the maximal cliques in an interval graph. For an interval graph with n intervals, let us denote the sorted endpoints of all the intervals as $e_1, e_2, \cdots, e_{2n}$. According to the theorem, *"all the intervals that cover an open interval $]e_s, e_{s+1}[$ form a maximal clique if and only if e_s is a left endpoint and e_{s+1} is a right endpoint, not necessarily of the same interval"*. Here, the left and the right endpoints imply the start and end of the intervals respectively. For example, the left and the right endpoints of an interval $]5, 8[$ are 5 and 8 respectively.

Consider an interval graph constructed for six open intervals as follows: (i) E $=]3, 9[$, (ii) S $=]8, 12[$, (iii) M $=]7, 11[$, (iv) C $=]4, 10[$, (v) H $=]2, 6[$, (vi) G $=]1, 5[$. The intervals are given in Figure 2.10(a) and the corresponding graph is as shown in Figure 2.9(a).. We can sort the endpoints of the intervals as follows:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.$$

Here, 4 and 5 are two consecutive endpoints where 4 is the left endpoint of C and 5 is the right endpoint of G. Thus, the interval $]4, 5[$ will correspond to a maximal clique. The intervals that contain $]4, 5[$ are E, C, H, and G, as shown in Figure 2.10(b). Similarly, 8 is the left endpoint of S and 9 is the right endpoint of E. $]4, 5[$ is covered by the intervals E, S, M, and C, as shown in Figure 2.10(b). Thus, there are two maximal cliques for this interval graph as can also be seen in Figure 2.9(a). Both maximal cliques have the same cardinality of four, thus, the number of colors used to color the interval graph is four as shown in Figure 2.9(b).

In Chapter 6, we will show that the properties of interval graphs can be exploited to formulate a closed-form optimization problem to minimize the balancing time for cell balancing.

3 Multi-Objective Co-Optimization for Distributed Cyber-Physical Systems[1]

3.1 Introduction

One emerging research direction towards an efficient design of cyber-physical systems (CPSs) is the *co-design* of the cyber platform and the controllers that will run on the platform. A co-design technique integrates the two design domains in an early phase of the development process, and therefore, the parameters on both sides can be computed according to certain design objectives. Such a technique exploits control-theoretic properties and platform-specific details to synthesize an optimal design configuration. In this chapter, we outline our proposed co-design methodology for distributed CPSs in the automotive context. Our proposed approach synthesizes feasible control and platform parameters, and at the same time, it also co-optimizes the average control performance and the resource usage respectively.

Distributed CPSs: Compared to the conventional notion of the embedded systems, CPS emphasizes more on the tight interaction between the computational elements and the physical entities. Here, we study embedded control systems, which are typical and important examples of CPSs. In such a system, the software implementation of a controller running on an embedded platform is used to control the physical plant. Such systems are often implemented on a distributed architecture where the control software is partitioned into multiple tasks mapped onto different processing units and the data between the tasks are transmitted on the bus. The design of embedded control systems mainly involves two aspects, namely the controller design and the platform design. The control parameters include, e.g., the control law and the sampling pe-

[1]This chapter is primarily based on a publication entitled "Multi-objective co-optimization of FlexRay-based distributed control systems" [195] that appeared at the 2016 IEEE Real-Time and Embedded Technology and Application Symposium (RTAS).

riod. The platform parameters refer usually to the design parameters of the underlying platform implementation, e.g., the task partition and mapping and the task and message schedules.

Design objectives for automotive systems: Distributed control systems are commonly found in the automotive domain. These systems are often safety-critical and have stringent performance requirements. The general idea is that the control performance improves with higher quality resources at higher quantity. Thus, to meet the safety requirements, high-quality resources are typically provisioned for control applications with significant conservativeness. That is, time-triggered resources are used at high frequency to execute software tasks and transmit communication messages for a safety-critical control application. On the other hand, the limitation of the resources is always one of the main constraints on an embedded platform in the automotive domain. First of all, the automotive industry is highly cost-sensitive and providing more resources is often coupled with increasing cost. Moreover, the scale and complexity of the hardware/software system in modern cars has increased drastically in the past decade. The resources are, therefore, becoming scarce, that makes resource-efficient design an important requirement for automotive systems. Note that the two objectives of maximizing control performance and minimizing resource usage are often conflicting to each other.

Control-platform co-design: In conventional design paradigm, the embedded platform and the controllers are designed separately and then integrated [196, 197]. Thus, control engineers make certain assumptions on control timings while designing the controllers, however, control timings depend on the platform implementation of the controllers. Similarly, platform parameters are also selected assuming that the control loops to be robust to a certain degree, however, the robustness of a control loop depends on the control law. If the assumptions made on control properties are too conservative, e.g., for safety-critical systems, the platform resources are usually not optimally utilized. On the other hand, if more idealistic assumptions are made on platform capabilities, the safety of the control loops can be jeopardized.

In recent years, the subject of control and platform co-design has emerged and has been drawing increasingly more attention [149, 161, 198]. This design paradigm combines the design of controllers and the underlying embedded platform together, often using a holistic approach, to explore the characteristics of both sides in order to reduce design conservativeness. In this design approach, the control parameters like control gains and sampling periods and the platform parameters like task and message schedules are computed by considering the interplay between the parameters. Often a mathematical model with platform-specific constraints and high-level control objectives is established and solved, which automates the design process. Thus, it offers the developer the opportunity to co-design control and platform parameters according to specific system-level objectives.

Challenges towards control-platform co-design: There still exist certain challenges towards a more comprehensive design of control and platform parameters for automotive systems. One challenge is the large design space. This is due to the fact that for various platform-specific behaviors such as sampling period, delay, and jitter, the control design problem does not fit into a standard closed-form optimal control framework. Therefore, it often becomes necessary to perform heuristic-based search to find the right system parameters such as poles. In addition, the platform requires the configuration of a huge number of parameters. A control application is implemented using several tasks and messages where the schedule of a task or a message comprises a few (two or three) configurable parameters, as discussed in Section 2.2. A combined space with controller and platform parameters is huge and challenging to handle. Moreover, in automotive systems, several applications share an ECU or a communication bus. Therefore, the control-platform co-design problem must consider control and platform parameters of all applications sharing the platform resources. Such a synthesis problem requires considerable computational effort. This is aggravated by the increase in the system size and the whole problem easily becomes intractable. Secondly, existing approaches usually consider only the optimization of the control performance as the design objective and there is no design freedom for trade-offs between the control performance and the resource usage.

Contributions: In our work, we have studied FlexRay based ECU networks, i.e., a number of ECUs are connected by a FlexRay bus. Such a FlexRay-based distributed platform is very common in the automotive domain for implementing safety-critical systems like steering and brake control. Towards control-platform co-design for distributed control systems, we make the following contributions:

- Compared to the existing approaches, we propose to co-optimize both average control performance and resource usage. Resource usage metric is formulated as the percentage of TDMA slots allocated to the control applications in the FlexRay static segment. Minimizing the resource usage would imply that slots are not conservatively reserved for control applications and unused slots can be used by other real-time data or for future applications, thus improving resource efficiency. On the control side, each application can be designed based on different control performance metrics (like settling time or quadratic cost) as per requirements. It must be noted here that control performance of different applications can be interdependent as these applications share limited platform resources. A single objective determining the average control performance of all applications is formulated to make the problem tractable. Thus, the individual performances are first normalized with respect to their required values. Average control performance is given by the weighted sum of the normalized performances. Now, for these objectives of resource usage and average control performance respectively, it is desirable to obtain a Pareto front depicting the trade-off.

- We integrate the problems of controller design and scheduling by accurately modeling the interplay between them. The sampling period and the delay with which a controller is designed depend on the task and message schedules. Properties of control theory and FlexRay protocol are exploited here to formulate the co-design problem efficiently. Typically, the impact of the closed-loop delay on the control performance is less as compared to the sampling period [199]. We assume that the delay is equal to one sampling period to offer more scheduling flexibility. In FlexRay, message cycle repetition rates can take only a limited number of values. The sampling period of a controller is equal to the period with which the corresponding tasks and messages are scheduled. Thus, sampling periods are also restricted to a predefined set of values corresponding to the feasible values of message repetition rates. Exploiting these aforementioned properties, we partition the co-design problem into two subsequent stages: (i) prospective controller design and (ii) co-optimization. Partitioning the whole problem into two stages is necessary because we are dealing with a large design space combining the dimensions of both control and platform designs. Thus, partitioning the whole design space into smaller sub-spaces while considering all feasible regions helps reducing the problem complexity. In the prospective controller design stage, optimal controllers are predesigned at all possible values of sampling period for each application. The controller design can, therefore, be integrated into the co-design problem where the solver in the co-optimization stage only needs to select one of these predesigned controllers. In addition, application-level constraints (like data dependency, sampling period and delay) and implementation rules (like non-conflicting resource allocations and finite resource capacities) are considered in the co-optimization stage. By predesigning the optimal controller at each possible value of sampling period, we avoid unnecessary schedule synthesis for sub-optimal or unstable controllers in the co-optimization stage, thus, enabling efficient design space pruning.

- The constrained multi-objective optimization problem considered in the second stage of the control-platform co-design cannot be solved efficiently by existing solvers. A combination of iterative search, MILP and integer linear programming (ILP) is applied to solve the problem. Note that only a finite set of values are possible for resource usage according to its definition. These values are traversed in ascending order. For a given value of resource usage, an optimization problem is formulated with average control performance as the only objective and an equality constraint on the resource usage. To ensure Pareto-optimality, an additional constraint is added, i.e., the objective value of the current solution must be the best among all the solutions obtained so far. Now, this single-objective optimization problem is solved in two nested layers to improve scalability. First, an MILP is solved with only sampling periods as variables and average control performance as the objective such that the equality constraint on the resource usage and the Pareto criterion are satisfied. Then, an ILP problem is solved to determine a valid set of task and message schedules satisfying the application-level and implementation-specific constraints. If a valid configuration is found then it represents a Pareto point. Otherwise, the next best set of sampling periods (still satisfying the Pareto criterion) is evaluated for feasibility.

- For a case study of only five control applications, 31 Pareto points are generated with a significant trade-off opportunity. That is, the percentage of TDMA slots used in the FlexRay static segment ranges from $5.5\,\%$ to $33\,\%$, while the average control performance varies between $40.6\,\%$ and $81.74\,\%$.[2] We have also evaluated the scalability of our approach, where the results show that this approach can scale up to a system size of at least 24 control applications.

Chapter organization: The rest of this chapter is organized as follows. In Section 3.2, we formulate the co-design problem based on the problem setting under study. Here, we also explain our proposed two-stage design flow. Section 3.3 describes the details of the first stage, i.e., the prospective controller design. In Section 3.4, we formulate the application- and platform-specific constraints for the multi-objective optimization in the second stage. Section 3.5 outlines the hybrid optimization technique that we have used in the second stage, i.e., the co-optimization. In Section 3.6, the experimental results based on a case study are presented, together with an analysis on the scalability of the proposed approach. Section 3.7 mentions the related works. Then, we provide the concluding remarks in Section 3.8.

[2]The required control performance of each application is considered as 100%.

3.2 Control-Platform Co-Design Problem

In this section, we will first study the problem setting where multiple physical plants are controlled by software running on a network of ECUs connected by a FlexRay bus. Based on the problem setting and the design objectives, we then define the co-design problem. Finally, we propose a design flow comprising two stages to solve the co-design problem.

3.2.1 Problem Setting

We consider a distributed architecture where a set of ECUs represented by $E_i \in \mathcal{E}$ are connected through a FlexRay bus. A number of control applications denoted by $\mathcal{C}_i \in \mathbb{C}$ are mapped on such an embedded platform. Each control application is partitioned into several dependent software tasks performing functions like sensor reading, computation and actuation. These tasks are typically mapped on physically distributed ECUs and the data between the tasks are sent via the FlexRay bus.

Task and message schedules: We consider the case where the scheduler used by the real-time operating system (RTOS) on the ECUs follows a time-triggered non-preemptive scheme. The tasks of the control applications are considered to be periodic tasks. Time-triggered periodic tasks are described in Section 2.2.1. The schedule of a task T_i is defined by the tuple $T_i \equiv \{o_i, p_i, e_i\}$, where o_i, p_i and e_i denote the offset, the period and the execution time of the task respectively. The start time $\widehat{t}(T_i, k)$ and the finish time $\widetilde{t}(T_i, k)$ of the $k-$th instance of a task T_i are given Eq. (2.35) and Eq. (2.36) respectively.

When two tasks need to communicate with each other while they are mapped on different ECUs, data can be sent as a message over the FlexRay bus. As stated in Section 2.2.2, the schedule of a FlexRay message m_i can be represented as a tuple $\{s_i, b_i, r_i\}$, where s_i, b_i, and r_i denote the slot id, the base cycle, and the repetition rate of the message. In this work, we assume that safety-critical control application uses the FlexRay static segment for communication. In the static segment, the start time $\widehat{t}(m_i, k)$ and the finish time $\widetilde{t}(m_i, k)$ of the $k-$th transmission of a message m_i is given by Eq. (2.39) and Eq.(2.40) respectively. We consider FlexRay 3.0.1, where slot multiplexing is possible without any restriction as mentioned in Section 2.2.2.

A set of *communication tasks* are required besides the application tasks. The communication task on the sending ECU writes the data produced by the application tasks into the corresponding transmit buffers of the communication controller, and on the receiving ECU, it reads the data from the corresponding receiver buffers and forwards them to the application tasks. The nature of these communication tasks depends on the specific implementation. Here we consider that the execution time of all communication tasks is bounded by e_{com} and we schedule the communication task directly after its corresponding application task at the sending side and directly before the application task at the receiving side.

Control applications: In this work, we consider linear and time-invariant (LTI) systems, for which the continuous-time plant dynamics can be written as in Eq. (2.1). Each controller $\mathcal{C}_i$ controlling a physical plant (defined by the continuous-time state-space matrices $\{A_i, B_i, C_i\}$) is implemented in a distributed fashion using tasks and messages. We assume that a task execution or a message transmission takes non-negligible time. Furthermore, based on the interplay

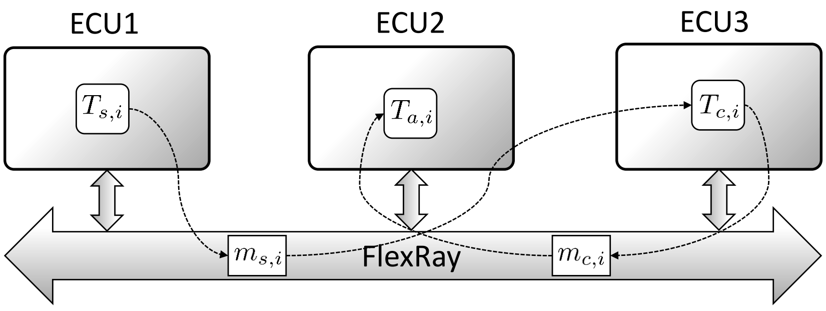


Figure 3.1: Distributed implementation of a controller. Each controller is implemented using three tasks mapped on to three different ECUs. Data between the tasks are sent via messages over the FlexRay bus.

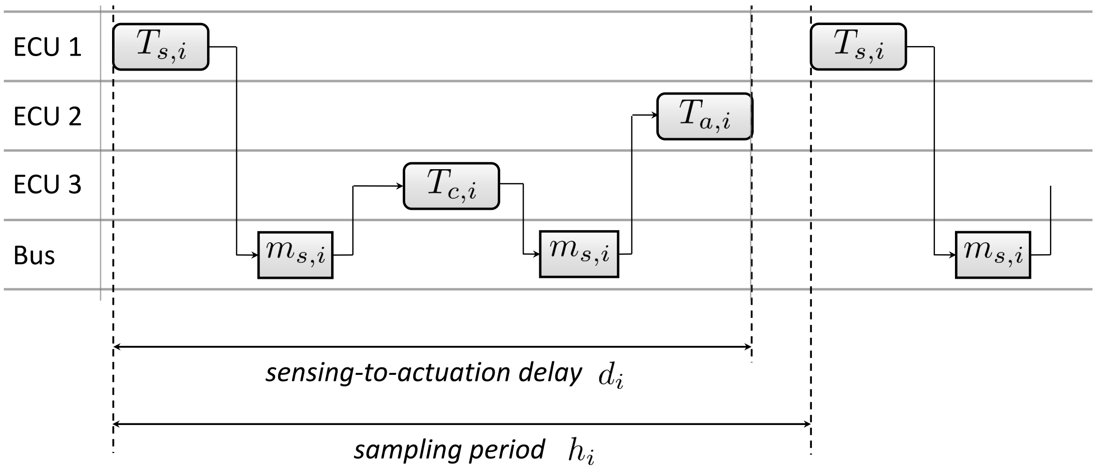


Figure 3.2: Control timings depend on the implementation. The sampling period and the sensing-to-actuation delay depend on the task and message schedules.

between the time-triggered schedules of tasks and messages constituting a control application, there will be a constant delay between the sensing and the actuation. For a sampling period h_i and a delay d_i, we can derive the delayed discrete-time state-space model as discussed in Section 2.1.3. Based on the augmented state-space model of delayed discrete-time systems as given in Eq. (2.16), we define the feedback control law as in Eq. (2.23). For such a definition of the controller, the feedback and feedforward control gains (i.e., K_i and F_i respectively) can be calculated by pole placement using Ackermann's formula as described in Section 2.1.5.

We consider that a control application is partitioned into three software tasks: (i) *sensor task* measures the system states (using sensors) of the physical system if measurable; (ii) *controller task* computes the controller input based on the measured system states; and (iii) *actuator task* applies the control input (using actuators) to the physical system. These software tasks are usually mapped onto different ECUs, due to the physically distributed topology of sensors and actuators. Here we denote the sensor, controller and actuator task of a control application $\mathcal{C}_i$ respectively as $T_{s,i}$, $T_{c,i}$ and $T_{a,i}$. Without loss of generality, we assume that the three tasks are mapped on different ECUs as shown in Figure 3.1. Then the sensor values measured by $T_{s,i}$ are sent on the bus via a message $m_{s,i}$ to the control task $T_{c,i}$. The control input is sent as a message $m_{c,i}$ by the control task $T_{c,i}$ to the actuator task $T_{a,i}$. The time between the start of the sensor task and the end of the actuator task is defined as the *sensing-to-actuation delay*, denoted as d_i. As shown in Figure 3.2, this delay depends on the the task and message schedules.

3.2.2 Problem Definition

For the problem setting described in Section 3.2.1, we will determine the control law and the task and message schedules for each control application, while co-optimizing the resource usage and the average control performance.

Resource usage: In this work, we minimize the usage of communication resource. This is because sufficient communication resource need to be provisioned for future applications. While adding a new application into a car, it might be possible to add an ECU if required but adding a new bus cluster is always challenging. Here, we define resource usage as the total amount of bandwidth of the FlexRay static segment that is allocated to the control applications. This can be translated to the number of static slots assigned to the control applications in a sequence of N_{com} communication cycles. Here, N_{com} is a power of 2 and it represents the number of configurable FlexRay communication cycles that repeats infinitely. In this work, we assume that N_{com} is predetermined.

Now, let $\mathcal{M}$ denote the set of all FlexRay messages mapped on the static segment, where $m_i \in \mathcal{M}$, then the resource usage U can be defined as follows:

$$U = \sum_{m_i \in \mathcal{M}} \frac{N_{com}}{r_i}, \tag{3.1}$$

where $\frac{N_{com}}{r_i}$ is the number of slots that the message m_i uses in N_{com} cycles The smaller the value of U, the lower is the resource usage and more resources can be provisioned for the future applications.

Note that the repetition rate of a message denotes how often the message is sent. For each application, the period of repetition of its constituent tasks and messages are equal. Thus, the metric U also gives an idea of how often the tasks are repeated. Thus, minimizing U will also minimize the usage of the ECU computation bandwidth.

Average control performance: There are different metrics to measure the performance of a control system. Here we consider two common metrics to measure the control performance. (i) The performance of a controller can be commonly measured by a *quadratic cost function* [161], which in the discrete-time control can be represented as

$$J = \sum_{k=0}^{n} [\lambda u[k]^2 + (1-\lambda)\sigma[k]^2]h, \tag{3.2}$$

where λ is a weight taking the value between 0 and 1, $u[k]$ is the control input and $\sigma[k] = |r - y[k]|$ is the tracking error. We further consider the (ii) *settling time* ξ as another control performance metric, where ξ denotes the time necessary for the system to reach and remain within a certain user-specified threshold of the reference value, i.e.,

$$J = \xi. \tag{3.3}$$

Depending on the specific requirements of the control application, one of the aforementioned performance metric can be used. In both cases, smaller value of J implies better control performance. For a specific control application $\mathcal{C}_i$, J_i depends on the sampling period h_i, the sensing-to-actuation delay d_i and the control gains K_i and F_i. In a system consisting of multiple control applications with different plant models and performance metrics, we need to normalize the control performance in order to compare and combine them. Each control system $\mathcal{C}_i$ with control performance J_i must satisfy some control performance requirement J_i^r defined by the user. Thus we can normalize it as follows:

$$J_i^n = \frac{100 \cdot J_i}{J_i^r} \tag{3.4}$$

and thus the average control performance of a set of control applications $\mathbb{C}$ is given by:

$$J_{av} = \frac{\sum_{\mathcal{C}_i \in \mathbb{C}} J_i^n}{|\mathbb{C}|}. \tag{3.5}$$

Co-optimization problem: Consider a control application $\mathcal{C}_i$ consisting of a sensor task $T_{s,i}$, a controller task $T_{c,i}$ and an actuator task $T_{a,i}$ and messages carrying the sensor data $m_{s,i}$ and the control input $m_{c,i}$. Each task $T_{tt,i}$ maps to a schedule $\{o_{tt,i}, p_{tt,i}, e_{tt,i}\}$, where tt is the task type and $tt \in \{s, c, a\}$. Note that we assume that the execution time is given for each task. Similarly, each message $m_{mt,i}$ maps to a FlexRay schedule $\{s_{mt,i}, b_{mt,i}, r_{mt,i}\}$, where mt is the message type and $mt \in \{s, c\}$. The control parameters for an application $\mathcal{C}_i$ include the sampling period h_i, the delay d_i, and the control gains K_i and F_i. Then the co-optimization problem boils down to finding a set of parameters for each $\mathcal{C}_i \in \mathbb{C}$, which can be denoted as par_i and is given by

$$par_i = \{o_{s,i}, p_{s,i}, o_{c,i}, p_{c,i}, o_{a,i}, p_{a,i}, s_{s,i}, b_{s,i}, r_{s,i}, s_{c,i}, b_{c,i}, r_{c,i}, h_i, d_i, K_i, F_i\}. \tag{3.6}$$

In the process, we also consider to optimize the communication resource usage U as in Eq. (3.1) and the average control performance J_{av} as in Eq. (3.5). Here, we further define the control parameters of $\mathcal{C}_i$ as $par_i^c = \{h_i, d_i, K_i, F_i\}$ and the embedded platform parameters as $par_i^s = \{o_{s,i}, p_{s,i}, o_{c,i}, p_{c,i}, o_{a,i}, p_{a,i}, s_{s,i}, b_{s,i}, r_{s,i}, s_{c,i}, b_{c,i}, r_{c,i}\}$, where $par_i = par_i^s \cup par_i^c$. The parameter set of the whole system is represented as $\mathcal{P}$, where $par_i \in \mathcal{P}$.

3.2.3 Proposed Design Flow

For the co-optimization problem defined in the last section, we propose an efficient two-stage approach as shown in Figure 3.3.

In the first stage, i.e., the prospective controller design stage, for each control application, prospective controllers are designed that optimize the control performance at different possible sampling periods and the results are recorded in a look-up table. As shown in Figure 3.3, for each application $\mathcal{C}_i$, the continuous-time plant model (A_i, B_i, C_i), the performance metric (i.e., settling time or quadratic cost) and the performance requirement (J_i^r) are taken as inputs. According to the FlexRay protocol, repetition rates can take only a limited number of values as given in Eq. (2.37). As the sampling period of a controller is equal to the period of repetition of the messages using which the controller is implemented, the choice of sampling

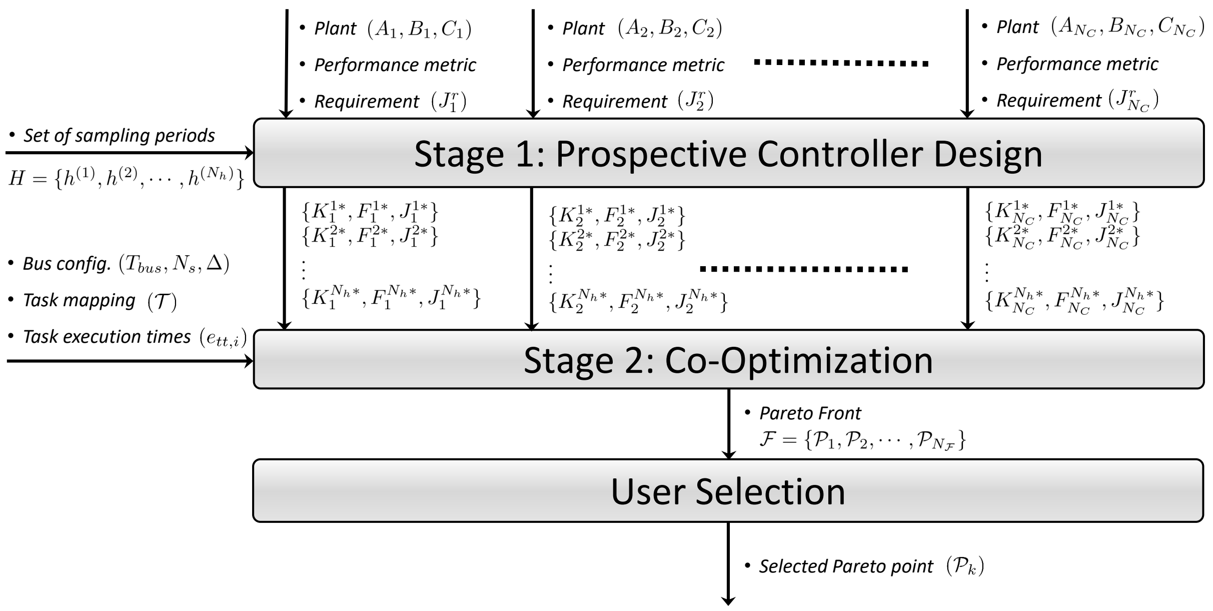

• Plant (A_1, B_1, C_1)
• Performance metric
• Requirement (J_1^r)
• Plant (A_2, B_2, C_2)
• Performance metric
• Requirement (J_2^r)
• Plant $(A_{N_C}, B_{N_C}, C_{N_C})$
• Performance metric
• Requirement $(J_{N_C}^r)$
• Set of sampling periods
$H = \{h^{(1)}, h^{(2)}, \cdots, h^{(N_h)}\}$
Stage 1: Prospective Controller Design
$\{K_1^{1*}, F_1^{1*}, J_1^{1*}\}$ $\{K_1^{2*}, F_1^{2*}, J_1^{2*}\}$ $\{K_1^{N_h*}, F_1^{N_h*}, J_1^{N_h*}\}$
$\{K_2^{1*}, F_2^{1*}, J_2^{1*}\}$ $\{K_2^{2*}, F_2^{2*}, J_2^{2*}\}$ $\{K_2^{N_h*}, F_2^{N_h*}, J_2^{N_h*}\}$
$\{K_{N_C}^{1*}, F_{N_C}^{1*}, J_{N_C}^{1*}\}$ $\{K_{N_C}^{2*}, F_{N_C}^{2*}, J_{N_C}^{2*}\}$ $\{K_{N_C}^{N_h*}, F_{N_C}^{N_h*}, J_{N_C}^{N_h*}\}$
• Bus config. (T_{bus}, N_s, Δ)
• Task mapping $(\mathcal{T})$
• Task execution times $(e_{tt,i})$
Stage 2: Co-Optimization
• Pareto Front
$\mathcal{F} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_{N_\mathcal{F}}\}$
User Selection
• Selected Pareto point $(\mathcal{P}_k)$

Figure 3.3: The proposed multi-stage control-platform co-design approach. The approach consists of two stages: (i) the prospective controller design and (ii) the co-optimization. In the first stage, prospective controller are designed for each application at each possible sampling period. In the second stage, Pareto-optimal design configurations are synthesized comprising control and platform parameters that respect all application-level and platform-specific constraints and co-optimize the resource usage and the average control performance respectively.

periods are also restricted correspondingly. The set of feasible sampling periods is denoted as $H = \{h^{(1)}, h^{(2)}, \cdots, h^{(N_h)}\}$. Corresponding to a sampling period $h^{(k)}$, we assume a delay equal to the sampling period, i.e., $d^{(k)} = h^{(k)}$, to allow more freedom to choose the task and message schedules. However, our approach can also be applied to any case where for each possible sampling period there is a constant delay. Now, for an application $\mathcal{C}_i \in \mathbb{C}$ and a sampling period $h_i = h^{(k)} \in H$, we design a prospective controller (K_i^{k*}, F_i^{k*}) that optimizes the control performance J_i assuming a sensing-to-actuation delay $d_i = d^{(k)}$. The optimal normalized control performance for an application $\mathcal{C}_i$ at a sampling period $h^{(k)}$ is denoted as J_i^{k*}. Thus, from the first stage, we obtain a table storing all prospective controllers and the corresponding normalized control performances for each application as shown in Figure 3.3. This stage will be explained in detail in Section 3.3.

In the second stage, the co-optimization stage, we synthesize feasible Pareto-optimal design configurations comprising both control and platform parameters. Besides the look-up table obtained from the first stage, this stage takes as inputs the task mapping $(\mathcal{T})^3$, execution times of the tasks implementing each control application $\{(e_{s,i}, e_{c,i}, e_{a,i}) | \mathcal{C}_i \in \mathbb{C}\}$, and certain FlexRay configuration parameters like bus cycle time (T_{bus}), number of static slots (N_S), and the length of a static slot (Δ). Using the input information, we mathematically formulate the application-level and platform-specific constraints and the design objectives for the co-optimization problem. We explain the constraints and the objectives and formulate them in Section 3.4. The

[3] $\mathcal{T}(E_j)$ denotes the set of tasks mapped on to the ECU E_j

co-optimization problem is solved using a customized hybrid optimization technique that generates the Pareto front for the two design objectives of resource usage and average control performance respectively. In this technique, Pareto point candidates are iteratively identified, exploring the characteristics of the resource usage objective, i.e., it can take only selected discrete values. For each candidate, a nested two-layer optimization finds a feasible set of control and platform parameters that optimize the control performance or proves that such a point is not feasible. The output of the second stage is a Pareto front representing optimal parameter sets taking both objectives into account. Based on this, the designer can select one set of parameters that is the most suitable for the overall design requirements. The co-optimization stage is dicussed in more detail in Section 3.5.

3.3 Stage 1: Prospective Controller Design

Controller design problem: Besides the control plant model, the control performance J_i of a control application $\mathcal{C}_i$ depends mainly on three factors: (i) the sampling period h_i, (ii) the sensing-to-actuation delay d_i and (iii) the control gains K_i and F_i. In this work, we consider that the schedules of the tasks and messages implementing a controller will lead to the case where the sensing-to-actuation delay is close to the sampling period, i.e., $d_i \approx h_i$. This would reduce the dimensions of the design space from all three factors (i) - (iii) to only (i) and (iii), thus reducing the complexity and enhancing the scalability. It should be noted that our approach is also valid for the cases with a fixed delay value (e.g., $d_i = D_i$, where D_i is a constant) or a delay value proportional to the sampling period (e.g. $d_i = \psi h_i$).

With $d_i = h_i$, the closed-loop system experiences one sampling period delay and we can use the pole placement method discussed in Section 2.1.5 for such a delayed system. This method enables to find the control gains according to a set of specified closed-loop poles for given values of sampling period and delay. For a discrete-time system, poles must be placed within a unit circle in the complex plane as discussed in Section 2.1.4. The design space for the controller design then consists of two dimensions, namely the sampling period and the closed-loop system poles. We can further make use of the fact the sampling period can only take discrete values to prune the design space. Each control application $\mathcal{C}_i$ is implemented using the tasks $T_{s,i}$, $T_{c,i}$, $T_{a,i}$ and the messages $m_{s,i}$, $m_{c,i}$, and therefore, there is a relation between the sampling period h_i and the repetition rate of the messages $r_{s,i}$, $r_{c,i}$. This relation can be written as follows:

$$h_i = r_{s,i} T_{bus} = r_{c,i} T_{bus}. \tag{3.7}$$

That is, the sampling period of a controller is equal to the period of repetition of the messages that are used for the controller implementation. Due to the fact that $r_{s,i}$, $r_{c,i}$ can only take discrete values in $\{2^{k-1} | k \in \{1, ..., 1 + \log_2 N_{com}\}\}$ as per Eq. (2.37), the choice of h_i is also constrained to the corresponding discrete values in the set $H = \{2^{k-1} \cdot T_{bus} | k \in \{1, ..., 1 + \log_2 N_{com}\}\}$. Thus, for each sampling period in the predefined set H, we need to determine the optimal set of poles that optimize the control performance. To the best of our knowledge, there is no standard closed-form optimal controller design framework for the performance metrics considered in this work. Therefore, we need to employ a heuristic-search based optimization to determine the optimal parameters.

We can denote the control performance as $J_i = f(h_i, K_i, F_i)$. Then, the control performance at each discrete value of the sampling period $h_i = 2^{k-1} \cdot T_{bus}$ can be represented as $J_i(k) = g_k(K_i, F_i)$. The purpose of the prospective controller design stage is to determine the control gains for each possible value of the sampling period that optimizes the control performance. Here, we search the space of the stable poles to determine a set of control gains K_i^{k*}, F_i^{k*}, that optimizes the control performance to J_i^{k*} at the sampling period $h_i = 2^{k-1} \cdot T_{bus}$.

Optimal controller design: In Section 2.1.5, we have discussed a pole placement technique that ensures closed-loop stability. However, it is challenging to derive a closed-form relation between pole locations and the control performance metrics considered here. Furthermore, we consider a constraint on control input while designing the controller, that makes the problem even more challenging. In almost every real-world system, due to physical constraints on the actuator, there is a limit on the control input (e.g., the maximum voltage or current that can be supplied by a battery) that can be applied to the plant. The controller needs to be designed such that the maximum value of the control input does not exceed this limit u^*, i.e., $\forall_{k\geq 0} u[k] \leq u^*$. There does not exist any closed-form optimal framework for such a controller design.

We can formulate the optimal pole placement problem as follows. The variables of the problem are the poles denoted as λ_γs. For the system to be stable the poles must lie within the unit circle in the complex plane, i.e., $|\lambda_\gamma| < 1$. The constraint on control input must be considered, i.e., $|u[k]| \leq u^*$. The objective of the problem is to minimize J_i. Note that J_i can represent the settling time or the quadratic cost as mentioned in Section 3.2.2, and in both cases, lower is the value of J_i, better the control performance is. In this work, we employ particle swarm optimization (PSO) to solve the optimal pole placement problem. PSO is discussed in detail in Section 2.4.2.

Algorithm 2 shows the optimal pole-placement algorithm using PSO. This algorithm determines the optimal controller for the application $\mathcal{C}_i$ at the sampling period $h_i = h^{(k)} = 2^{k-1} \cdot T_{bus}$. It takes as inputs the augmented state-space system matrices ($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}$), the sampling period $h^{(k)}$, the delay $d^{(k)}$, and the maximum permissible value of the control input (u^*). It returns the optimal feedback and feedforward control gains (K_i^{k*} and F_i^{k*}) and the corresponding optimal control performance (J_i^{k*}).

The algorithm starts by finding the number of variables ($NumPoles$) of the optimization problem, i.e., the number of poles to be placed (line 1). Here, the function **CountStates**$(\cdot)$ determines the number of states of the system from the dimensions of the state transition matrix $\phi_{a,i}$. Let us consider that the number of particles that we use to search the pole space be $ParNum$. For each particle, the algorithm tries to find a starting position that is a valid solution (lines 2-16). Correspondingly, the following steps are carried out in order. (i) First, the poles represented by the position of the particle are assigned random real values within the unit circle while the velocities are initialized as 0 (lines 4-7). Here, **rnd**(-1,1) returns random real numbers between -1 and 1. (ii) In line 8, the algorithm finds the feedback and feedforward gains by placing the poles that are randomly assigned. (iii) For the calculated values of control gains, the system is simulated to get the response and the input curves (line 9). (iv) We can calculate the fitness value, i.e., the control performance from the result of the simulation using the function **FindPerformance**$(\cdot)$ (line 10). (v) We find the maximum value of the control input from the input vector $\overline{u}$ using the function **FindMaximumInput**$(\cdot)$ (line 11). (vi) If the constraint on the control input is respected then the randomly assigned poles represent a valid solution and

Algorithm 2: Optimal pole-placement using particle swarm optimization

```
   Inputs      : $\{\phi_{a,i}, \Gamma_{a,i}, C_{a,i}\}, h^{(k)}, d^{(k)}, u^*$
   Outputs     : $K_i^{k*}, F_i^{k*}, J_i^{k*}$
   Parameters  : $IterNum$, $ParNum$, $\{\alpha_0, \alpha_1, \alpha_2\}$
1  $NumPoles =$ CountStates($\phi_{a,i}$);
2  $pt = 1$;
3  while $pt \leq ParNum$ do
4      for $np = 1$ to $NumPoles$ do
5          $pos_{pt}[np] =$ rnd($-1, 1$);
6          $vel_{pt}[np] = 0$;
7      end
8      $[K, F] =$ PolePlacement($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}, pos_{pt}$);
9      $[\overline{y}, \overline{u}] =$ Simulate($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}, K, F, h^{(k)}, d^{(k)}$);
10     $fit_{pt} =$ FindPerformance($\overline{y}, \overline{u}, h^{(k)}, d^{(k)}$);
11     $\widehat{u} =$ FindMaximumInput($\overline{u}$);
12     if $\widehat{u} \leq u^*$ then
13         $pbest_{pt} = \{pos_{pt}, fit_{pt}, \widehat{u}\}$;
14         $pt = pt + 1$;
15     end
16 end
17 $gbest =$ FindBestSolution($\{pbest_{pt}\}$);
18 for $in = 1$ to $IterNum$ do
19     for $pt = 1$ to $ParNum$ do
20         $vel_{pt} = \alpha_0 \cdot vel_{pt} + \alpha_1 \cdot$ rnd$(0, 1) \cdot (pbest_{pt} - pos_{pt}) + \alpha_2 \cdot$ rnd$(0, 1) \cdot (gbest - pos_{pt})$;
21         $pos_{pt} = pos_{pt} + vel_{pt}$;
22         $[K, F] =$ PolePlacement($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}, pos_{pt}$);
23         $[\overline{y}, \overline{u}] =$ Simulate($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}, K, F, h^{(k)}, d^{(k)}$);
24         $fit_{pt} =$ FindPerformance($\overline{y}, \overline{u}, h^{(k)}, d^{(k)}$);
25         $\widehat{u} =$ FindMaximumInput($\overline{u}$);
26         $pbest_{pt} =$ FindBetterSolution($pbest_{pt}, \{pos_{pt}, fit_{pt}, \widehat{u}\}$);
27     end
28     $gbest =$ FindBestSolution($\{pbest_{pt}\}$);
29 end
30 $[K_i^{k*}, F_i^{k*}] =$ PolePlacement($\phi_{a,i}, \Gamma_{a,i}, C_{a,i}, gbest.pos$);
31 $J_i^{k*} = gbest.fit$;
32 return $K_i^{k*}, F_i^{k*}, J_i^{k*}$;
```

can be used as the starting position otherwise we try a new assignment (lines 12-15). (vii) The starting position of a particle is also the personal best solution achieved so far by the particle (line 14). After all particles are initialized, $gbest$ can be calculated as the best solution among all the particles using the function **FindBestSolution**($\cdots$) (line 16). Note that two solution can be compared using the rules described in Section 2.4.2 to determine the better one.

Now, the next positions of all the particles are calculated iteratively until a finite number of iterations $IterNum$ (lines 18-29). In each iteration, for each particle, we first calculate the new velocity and position using Eq. (2.54) and Eq. (2.55) respectively (lines 20-21). Corresponding to the new position, we can again calculate the fitness value and the maximum input applied (lines 22-25). Then, we can update the personal best of a particle by comparing the newly evaluated point to the old personal best solution (line 26). At the end of each iteration we must update $gbest$ with the best solution among all the personal best solutions (line 28). After $IterNum$ number of iterations, the algorithm calculates the optimal gains (K_i^{k*} and F_i^{k*}) using the position of the $gbest$ solution (line 30) and the optimal control performance (J_i^{k*}) is the fitness value of the $gbest$ solution (line 31). Finally, the algorithm returns the optimal controller and the optimal control performance for the application $\mathcal{C}_i$ corresponding to the sampling period $h^{(k)}$ and the delay $d^{(k)}$ (line 32).

3.4 Stage 2: Constrained Optimization Problem

Using the results of the prospective controller design stage, we can further formulate a constrained optimization problem as will be discussed in this section. The parameters to be synthesized here include (i) the sampling period of the control applications, (ii) the task schedules and (iii) the message schedules. Note that given a sampling period of an application, we can look-up the optimal control gains from the table obtained from the first stage. The optimization problem formulated here consists of the platform constraints and the control performance constraints. As the optimization objectives, we consider the resource usage and the average control performance respectively.

3.4.1 Constraints

The scheduling constraints for FlexRay-based ECU networks are well-studied and discussed in [161, 200, 201]. We state the constraints specific to the problem under study as follows:

(C1) Sampling period constraint: As mentioned earlier, all tasks and messages constituting a control application must have the same period of repetition that must be equal to the sampling period of the control loop. This constraint can be formulated as follows:

$$\forall \mathcal{C}_i \in \mathbb{C}, tt \in \{s,c,a\}, mt \in \{s,c\}, \qquad p_{tt,i} = r_{mt,i} \cdot T_{bus} = h_i. \tag{3.8}$$

That is, the periods of the sensor, controller and actuator tasks must be equal to the sampling period. The repetition rates of the sensor and control messages must be equal to $\frac{h_i}{T_{bus}}$.

(C2) Dataflow constraint: In a control application all task executions and message transmission have to be carried out in the correct temporal order, as illustrated in Figure 3.2. Correspondingly, we can derive the following relations:

- *Relation between the sensor task and the sensor message:* The sensor data must be made available at the sending buffer by the communication task (following the sensor task) before the sensor message is sent. This can be formulated as follows:

$$\begin{aligned} &\forall k \in \mathbb{Z}^+, \mathcal{C}_i \in \mathbb{C}, \quad \tilde{t}(T_{s,i}, k) + e_{com} < \widehat{t}(m_{s,i}, k) + \beta_{s,i} \cdot r_{s,i} \cdot T_{bus} \\ \Longrightarrow &\forall \mathcal{C}_i \in \mathbb{C}, \quad o_{s,i} + e_{s,i} + e_{com} < b_{s,i} \cdot T_{bus} + (s_{s,i} - 1) \cdot \Delta + \beta_{s,i} \cdot r_{s,i} \cdot T_{bus}. \\ &[\because \text{ from Eq. (2.36) and Eq. (2.39)}] \end{aligned} \tag{3.9}$$

Here, $\beta_{s,i} \in \{0, 1\}$. Note that the $k-$th instance of the sensor task can be followed by the $(k+1)-$th instance of the sensor message without violating the dataflow constraint. This is considered in this constraint by adding the term $\beta_{s,i} \cdot r_{s,i} \cdot T_{bus}$ to the right side of the inequality.

- *Relation between the sensor message and the controller task:* The sensor message must reach the receiving buffer before the communication task (preceding the controller task) starts reading the data. This constraint can be formulated as follows:

$$\begin{aligned} &\forall k \in \mathbb{Z}^+, \mathcal{C}_i \in \mathbb{C}, \quad \tilde{t}(m_{s,i}, k) + \beta_{s,i} \cdot r_{s,i} \cdot T_{bus} < \widehat{t}(T_{c,i}, k) - e_{com} + \alpha_{c,i} \cdot p_{c,i} \\ \Longrightarrow &\forall \mathcal{C}_i \in \mathbb{C}, \quad b_{s,i} \cdot T_{bus} + s_{s,i} \cdot \Delta + \beta_{s,i} \cdot r_{s,i} \cdot T_{bus} < o_{c,i} - e_{com} + \alpha_{c,i} p_{c,i}. \\ &[\because \text{ from Eq. (2.40) and Eq. (2.35)}] \end{aligned} \tag{3.10}$$

Here, $\alpha_{c,i} \in \{0, 1\}$.

- *Relation between the controller task and control message:* The control input must be made available at the sending buffer by the communication task (following the controller task) before the control message is sent. This constraint can be formulated as follows:

$$\begin{aligned} &\forall k \in \mathbb{Z}^+, \mathcal{C}_i \in \mathbb{C}, \quad \tilde{t}(T_{c,i}, k) + e_{com} + \alpha_{c,i} \cdot p_{c,i} < \widehat{t}(m_{c,i}, k) + \beta_{c,i} \cdot r_{c,i} \cdot T_{bus} \\ \Longrightarrow &\forall \mathcal{C}_i \in \mathbb{C}, \ o_{c,i} + e_{c,i} + e_{com} + \alpha_{c,i} \cdot p_{c,i} < b_{c,i} \cdot T_{bus} + (s_{c,i} - 1) \cdot \Delta + \beta_{c,i} \cdot r_{c,i} \cdot T_{bus} \\ &[\because \text{ from Eq. (2.36) and Eq. (2.39)}] \end{aligned} \tag{3.11}$$

Here, $\beta_{c,i} \in \{0, 1\}$.

- *Relation between the control message and the actuator task:* The control message must reach the receiving buffer before the communication task (preceding the actuator task) starts reading the data. This constraint can be formulated as follows:

$$\begin{aligned} &\forall k \in \mathbb{Z}^+, \mathcal{C}_i \in \mathbb{C}, \quad \tilde{t}(m_{c,i}, k) + \beta_{c,i} \cdot r_{c,i} \cdot T_{bus} < \widehat{t}(T_{a,i}, k) - e_{com} + \alpha_{a,i} p_{a,i} \\ \Longrightarrow &\forall \mathcal{C}_i \in \mathbb{C}, \quad b_{c,i} \cdot T_{bus} + s_{c,i} \cdot \Delta + \beta_{c,i} \cdot r_{c,i} \cdot T_{bus} < o_{a,i} - e_{com} + \alpha_{a,i} \cdot p_{a,i}. \\ &[\because \text{ from Eq. (2.40) and Eq. (2.35)}] \end{aligned} \tag{3.12}$$

Here, $\alpha_{a,i} \in \{0, 1\}$.

(C3) Sensing-to-actuation delay constraint: This constraint states that the start time of the sensor task and the end time of the actuator task must be separated by a time equal to the sensing-to-actuation delay. This is also illustrated in Figure 3.2. This constraint can be formulated as follows:

$$\begin{aligned} &\forall k \in \mathbb{Z}^+, \mathcal{C}_i \in \mathbb{C}, \quad \tilde{t}(\tau_{a,i}, k) + \alpha_{a,i} \cdot p_{a,i} - \widehat{t}(\tau_{s,i}, k) = d_i \\ \Longrightarrow &\forall \mathcal{C}_i \in \mathbb{C}, \quad (o_{a,i} + e_{a,i} + \alpha_{a,i} \cdot p_{a,i}) - o_{s,i} = d_i. \ [\because \text{from Eq. (2.35) and Eq. (2.36)}] \end{aligned} \tag{3.13}$$

(C4) Non-overlapping tasks constraint: In a time-triggered non-preemptive scheduling scheme as considered in this work, when more than one task is mapped on an ECU, they must be scheduled in such a way that they do not overlap. This constraint can be formulated as follows:

$$\begin{gathered} \forall \quad \mathcal{C}_i, C_j \in \mathbb{C}, \quad tt, tt' \in \{s, c, a\}, \quad E_k \in \mathcal{E} \\ \forall \quad \{k_1 \in \mathbb{Z}^+ | 1 \leq k_1 \leq \frac{lcm(p_{tt,i}, p_{tt',j})}{p_{tt,i}}\}, \quad \{k_2 \in \mathbb{Z}^+ | 1 \leq k_2 \leq \frac{lcm(p_{tt,i}, p_{tt',j})}{p_{tt',j}}\} \\ \textbf{if} \quad T_{tt,i}, T_{tt',j} \in \mathcal{T}(E_k) \quad \textbf{then} \\ \tilde{t}(\tau_{tt,i}, k_1) + e_{com} \cdot \mathbb{1}(tt \in \{s, c\}) < \widehat{t}(T_{tt',j}, k_2) - e_{com} \cdot \mathbb{1}(tt' \in \{c, a\}) \\ \textbf{or} \\ \tilde{t}(T_{tt',j}, k_1) + e_{com} \cdot \mathbb{1}(tt' \in \{s, c\}) < \widehat{t}(T_{tt,i}, k_2) - e_{com} \cdot \mathbb{1}(tt \in \{c, a\}), \end{gathered}$$

where $\mathcal{T}(E_k)$ denotes the set of all tasks mapped on the ECU E_k. $\mathbb{1}(.)$ is the indicator function and takes the value of 1 if the input is true and 0 if otherwise. From Eq. (2.35) and Eq. (2.36), we can rewrite the constraint as follows:

$$\begin{gathered} \forall \quad \mathcal{C}_i, C_j \in \mathbb{C}, \quad tt, tt' \in \{s, c, a\}, \quad E_k \in \mathcal{E} \\ \forall \quad \{k_1 \in \mathbb{Z}^+ | 1 \leq k_1 \leq \frac{lcm(p_{tt,i}, p_{tt',j})}{p_{tt,i}}\}, \quad \{k_2 \in \mathbb{Z}^+ | 1 \leq k_2 \leq \frac{lcm(p_{tt,i}, p_{tt',j})}{p_{tt',j}}\} \\ \textbf{if} \quad T_{tt,i}, T_{tt',j} \in \mathcal{T}(E_k) \quad \textbf{then} \\ o_{tt,i} + e_{tt,i} + k_1 \cdot p_{tt,i} + e_{com} \cdot \mathbb{1}(tt \in \{s, c\}) < o_{tt',j} + k_2 \cdot p_{tt',j} - e_{com} \cdot \mathbb{1}(tt' \in \{c, a\}) \\ \textbf{or} \\ o_{tt',j} + e_{tt',j} + k_2 \cdot p_{tt',j} + e_{com} \cdot \mathbb{1}(tt' \in \{s, c\}) < o_{tt,i} + k_1 \cdot p_{tt,i} - e_{com} \cdot \mathbb{1}(tt \in \{c, a\}). \end{gathered} \tag{3.14}$$

(C5) Non-overlapping messages constraint: FlexRay messages must be scheduled in such a way that no two messages share the same slot in the same cycle. This constraint can be established as follows:

$$\begin{gathered} \forall \quad \mathcal{C}_i, C_j \in \mathbb{C}, \quad mt, mt' \in \{s, c\} \\ \forall \{k_1 \in \mathbb{Z}^+ | 1 \leq k_1 \leq \frac{max(R_{mt,i}, R_{mt',j})}{R_{mt,i}}\}, \quad \{k_2 \in \mathbb{Z}^+ | 0 \leq k_2 < \frac{max(R_{mt,i}, R_{mt',j})}{R_{mt',j}}\} \\ \textbf{if} \quad s_{mt,i} == s_{mt',j} \quad \textbf{then} \quad b_{mt,i} + k_1 \cdot r_{mt,i} \neq b_{mt',j} + k_2 \cdot r_{mt',j}. \end{gathered} \tag{3.15}$$

(C6) FlexRay scheduling constraint: Taking into consideration the scheduling constraints imposed by the FlexRay protocol as discussed in Section 2.2.2, we need to constrain the repetition rates ($r_{s,i}$ and $r_{s,i}$) and the base cycles ($b_{s,i}$ and $b_{s,i}$) as per Eq. (2.37) and Eq. (2.38) respectively. Furthermore, we consider scheduling FlexRay messages only in the static segment, and therefore, the slot ids ($s_{s,i}$ and $s_{c,i}$) can take values as follows:

$$\forall \;\; \mathcal{C}_i \in \mathbb{C}, \quad mt \in \{s, c\}, \quad 1 \leq s_{mt,i} \leq N_s.^{4} \tag{3.16}$$

In addition, the resource usage as defined in Eq. (3.1) must not exceed the total number of static slots available in N_{com} communication cycles. This can be written as follows:

$$U \leq N_{com} \cdot N_s. \tag{3.17}$$

(C7) ECU scheduling constraint: On the ECUs, for the task schedules, we consider the following constraint:

$$\begin{gathered} \forall \;\; \mathcal{C}_i \in \mathbb{C}, \;\; tt \in \{s, c, a\}, \\ e_{com} \cdot \mathbb{1}(tt \in \{c, a\}) \leq o_{tt,i} < p_{tt,i} - e_{com} \cdot \mathbb{1}(tt \in \{s, c\}) - e_{tt,i} - \gamma_{syn}, \end{gathered} \tag{3.18}$$

where, γ_{syn} is a time constant required for ECU synchronization.

The utilization constraint on an ECU can be formulated as follows:

$$\forall E_k \in \mathcal{E}, tt \in \{s, c, a\}, \sum_{T_{tt,i} \in \mathcal{T}(E_k)} \frac{e_{tt,i} + e_{com} + e_{com} \cdot \mathbb{1}(tt \in \{c\})}{p_{tt,i}} \leq 1. \tag{3.19}$$

This constraint states that an ECU cannot be more than 100% loaded.

(C8) Performance constraint: For each application $\mathcal{C}_i$, the sampling period h_i is restricted to a discrete set of values, which can be written as follows:

$$h_i = \sum_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} \cdot 2^{k-1} \cdot T_{bus}, \quad \text{where} \quad \sum_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} = 1. \tag{3.20}$$

That is, only one of the boolean variables $\psi_{i,k}$s is 1, and the corresponding value of sampling period is selected.

In the prospective controller design stage as described in Section 3.3, we have obtained the optimal control gains and the corresponding normalized control performance for each application at all possible sampling periods. In this stage, the design space consists of only the predesigned controllers from the first stage. Thus, the normalized control performance J_i^n of an application can be written as:

$$J_i = \sum_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} J_i^{k*}. \tag{3.21}$$

[4] N_s is the number of static slots in a FlexRay communication cycle.

That is, corresponding to choosing a sampling period $h_i = h^{(k)} = 2^{k-1} \cdot T_{bus}$, the normalized control performance attains the value J_i^{K*}.

Note that each application $\mathcal{C}_i$ must satisfy a control performance requirement J_i^r, corresponding to which the normalized value is 100%. The performance constraints can, therefore, be written as follows:

$$J_i^{k*} > 100\% \implies \psi_{i,k} = 0. \tag{3.22}$$

That is, if the optimal normalized control performance J_i^{k*} corresponding to a sampling period $h^{(k)}$ is greater than 100% (i.e., requirement is violated), then $h_i \neq h^{(k)}$ or the corresponding boolean variable $\psi_{i,k}$ is equal to 0.

3.4.2 Optimization Objectives

As the objectives of the optimization problem, we consider the average control performance and the resource usage.

(O1) Average control performance: Given the formulation of control performance in Eq. (3.21), we can reformulate the objective of average control performance from Eq. (3.5) as follows:

$$J_{av} = \frac{\sum\limits_{i=1}^{|\mathbb{C}|} \sum\limits_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} J_i^{k*}}{|\mathbb{C}|}. \tag{3.23}$$

Thus, from the look-up table obtained in the prospective controller design stage, we can formulate the objective of resource usage.

(O2) Resource usage: Based on the problem setting and Eq. (3.8), we can reformulate the objective of resource usage from Eq. (3.1) as follows:

$$U = \sum_{\mathcal{C}_i \in \mathbb{C}} \left(\frac{64}{r_{s,i}} + \frac{64}{r_{c,i}} \right) = \sum_{\mathcal{C}_i \in \mathbb{C}} \frac{128 T_{bus}}{h_i}. \tag{3.24}$$

The value of the resources usage can only take certain discrete values and is bounded by the upper and the lower limit (U^+ and U^-), which can be derived respectively as follows:

$$\begin{aligned} U^+ &= \sum_{\mathcal{C}_i \in \mathbb{C}} \frac{128 T_{bus}}{\min\limits_{h^{(k)} \in dom(h_i)} h^{(k)}}, \\ U^- &= \sum_{\mathcal{C}_i \in \mathbb{C}} \frac{128 T_{bus}}{\max\limits_{h^{(k)} \in dom(h_i)} h^{(k)}}, \end{aligned} \tag{3.25}$$

where $dom(h_i)$, for each application $\mathcal{C}_i$, is given by:

$$dom(h_i) = \{h^{(k)} \in H | J_i^{k*} \leq 100\%\}. \tag{3.26}$$

Furthermore, we can also determine the minimum difference U^Δ between two values of resource usage as follows:

$$U^\Delta = \frac{128 T_{bus}}{\max\limits_{\mathcal{C}_i \in \mathbb{C}} \left(\max\limits_{h^{(k)} \in dom(h_i)} h^{(k)} \right)}. \tag{3.27}$$

3.5 Stage 2: Proposed Multi-Layer Hybrid Optimization

As discussed in Section 3.4, the control and schedule co-synthesis for the problem setting under study can be formulated as a constrained optimization problem with two objectives, namely the resource usage and the average control performance respectively. There exist several methods dealing with multi-objective optimization. A simple way is to convert the multiple objectives into one single objective with scalarization. However, the problems using this method here are (i) both objectives are completely different in nature and difficult to be combined as a single metric, and (ii) it would not be possible for the designer to fathom the design trade-off, which is necessary since the two design objectives are noticed to be often conflicting. In this case, a much more informative and designer-friendly approach is to first generate a Pareto front and let the designer explore the trade-off between the two objectives according to his/her customized preference.

In computing the Pareto points forming up the Pareto front, there are two requirements. Firstly, the obtained design points cannot be dominated by any other point in the objective space. In other words, there can be no point that is better than the solution points in both objectives. Secondly, the Pareto front should have a good space distribution. Design points too close to each other are of little help. It is noted that a Pareto point can be obtained by assigning a weight to each objective depending on its importance and optimizing the sum of all objectives. By changing the set of weights, different solutions can be obtained after solving the single-objective optimization problem. With this method, the first requirement on dominance is guaranteed. However, well distributed weights do not necessarily generate well distributed Pareto points, which means that the second requirement on distribution might not be fulfilled.

In this work, we propose a customized optimization approach to obtain the desired Pareto front. Our approach consists of three nested layers as shown in Figure 3.4. The objective of resource usage U is discrete and only takes a limited number of integers. We iterate through all feasible values of U in the outer layer. For a given value of U, we solve an optimization problem with average control performance J_{av} as the only objective and an equality constraint on U. This single objective optimization problem is solved in the inner two layers of the proposed optimization approach. Here, the objectives of resource usage and average control performance depend only on the values of sampling periods. Thus, in the middle layer, we compute the values of sampling periods $\{h_i | \mathcal{C}_i \in \mathbb{C}\}$ that optimize J_{av} for a given value of U. Now, in the innermost layer, we compute the schedules according to the obtained values of sampling periods from the middle layer while respecting the scheduling constraints.

3.5.1 Outer Layer

Since the objective of resource usage U is discrete, we first compute the maximum and minimum resource usage U^+ and U^-, that bound the set of U. For each possible value of U between U^- and U^+, i.e., given the equality constraint on U, we solve the optimization problem with J_{av} as the single objective and obtain a solution. The additional constraint is that J_{av} of this solution has to be better than J_{av} of the last solution, in order to ensure that all solutions are non-dominated, which corresponds to the first requirement of the Pareto points. With this method, the obtained Pareto points have a good distribution on the resource usage, since for each possi-

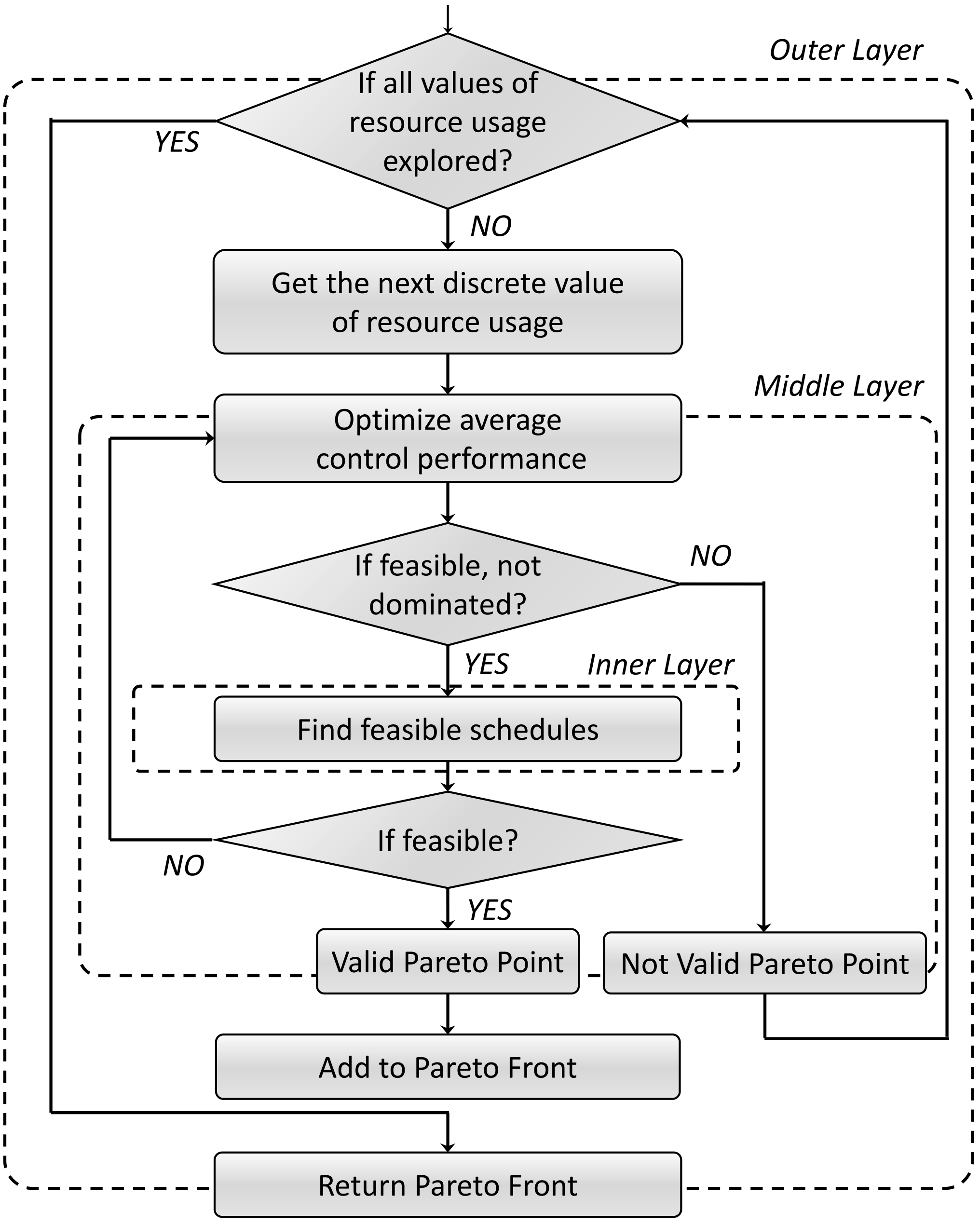


Figure 3.4: The proposed multi-layer co-optimization of resource usage and average control performance. In the outer layer, the discrete values of resource usage are explore one by one. In the middle layer, the values of sampling periods are computed that optimize the average control performance for the given value of resource usage. In the inner layer, schedules are synthesized according for given values of sampling periods.

ble value of U, there is a solution generated, as long as there exists a non-dominated solution. This fulfills the second requirement of the Pareto points.

Algorithm 3 shows the outer layer of the proposed multi-layer optimization approach. The algorithm takes as inputs the feasible set of sampling periods H and a table $\mathbb{J}$ comprising the

Algorithm 3: The outer layer.

```
Inputs          : $\{2^{k-1} \cdot T_{bus} \in H | 1 \leq k \leq \log_2 N_{com}\}$,
                  $\{J_i^{k*} \in \mathbb{J} | \mathcal{C}_i \in \mathbb{C} \wedge 2^{k-1} \cdot T_{bus} \in H\}$
Output          : $\mathcal{F}$
Initializations : $\mathcal{F} = \{\}$, $J^+ = \infty$
1  $U^- =$ calculateMinU();
2  $U^+ =$ calculateMaxU();
3  $U^\Delta =$ calculateMinDiff();
4  for $U^* \leftarrow U^-$ to $U^+$ by $U^\Delta$ do
5      $[\mathcal{P}^*, isPareto, J_{av}^*] =$ findPareto($U^*, J^+$);
6      if $true == isPareto$ then
7          $J^+ = J_{av}^*$;
8          $pp^* = [U^*, J_{av}^*, \mathcal{P}^*]$;
9          $\mathcal{F}$.addPareto($pp^*$);
10     end
11 end
12 return $\mathcal{F}$;
```

optimal control performance value at each feasible sampling period for each application (as obtained in the prospective controller design stage). Here we denote a Pareto point as $pp^* = \{U^*, J_{av}^*, \mathcal{P}^*\}$, where U^* stands for a discrete value of U, and J_{av}^* and $\mathcal{P}^*$ represent respectively the corresponding optimal average control performance and parameter set. The Pareto front is denoted as $\mathcal{F} = \{pp^*\}$ which is the output of the algorithm. In Algorithm 3, the Pareto front $\mathcal{F}$ is initialized as an empty set and J^+ as infinity. Lines 1-2 calculate the minimal and the maximal value of U according to Eq. (3.25) and Eq. (3.26). Line 3 calculates the minimum difference between two discrete values of resource usage as per Eq. (3.27) and Eq. (3.26). Then, in the *for loop* between line 4 and line 11, the algorithm traverses all possible values of U. For each discrete value of resource usage U^*, the inner layers of the optimization (line 5) is called to find a feasible parameter set that optimizes the average control performance and can be considered as a Pareto point (i.e. it is not dominated by other points). The middle and the inner layers is described next in Section 3.5.2. If the feasible parameter set can be obtained, it will be added to the Pareto front $\mathcal{F}$ (lines 6 - 10). Note that here J^+ stores the best obtained value of average control performance so far in the algorithm, and it is used in the middle layer to ensure the Pareto criterion for the next obtained design configuration.

3.5.2 Middle and Inner Layers

As already discussed in the previous section, the co-optimization problem with two objectives is turned into a series of single-objective optimization problems, where each may generate a Pareto point on the Pareto front. Popular approaches for solving such optimization problems include MILP or meta-heuristic methods. However, MILP is not applicable in this case. This is due to two facts. First, some constraints are not linear and cannot be linearized in a straightforward way. Second, formulating the whole problem into one single model would cause the number of variables and constraints to explode, thus making it extremely computationally expensive and not solvable for larger system sizes. On the other hand, although meta-heuristic methods like evolutionary algorithms are often applied to deal with high complexity, their capability to handle constraints is often limited. As the design space of the optimization problem formulated here is highly constrained, it is difficult for meta-heuristics to find a solution that respects all constraints, and this prevents meta-heuristics from being effective.

Algorithm 4: findPareto: The middle and the inner layers.

```
Inputs          : $U^*$, $J^+$
Outputs         : $\mathcal{P}^*$, $isPareto$, $J^*_{av}$
Initializations : $isPareto = false$, $J^- = 0$
while $false == isPareto$ do
    $[\{\mathcal{P}^c\}, J^*_{av}, isFeasibleU]$ = optimizeCP($U^*, J^-, J^+$);
    if $false == isFeasibleU$ then
        break;
    else
        $J^- = J^*_{av}$;
        $\mathcal{P}^{c*} = \{\mathcal{P}^c\}$.getFirstSet();
        while $\mathcal{P}^{c*} \neq \{\}$ and $false == isPareto$ do
            $[\mathcal{P}^{s*}, isFeasibleH]$ = findSchedules($\mathcal{P}^{c*}$);
            if $true == isFeasibleH$ then
                $\mathcal{P}^* = \mathcal{P}^{c*} \cup \mathcal{P}^{s*}$;
                $isPareto = true$;
                break;
            else
                $\mathcal{P}^{c*} = \{\mathcal{P}^c\}$.getNextSet();
            end
        end
    end
end
if $true == isPareto$ then
    return $\mathcal{P}^*$, $J^*_{av}$, $isPareto$;
else
    return $\mathcal{P}^* = \emptyset$, $J^*_{av} = 0$, $isPareto$;
end
```

Considering that some decision variables only appear in constraints, but are not related to the objective, we propose to solve the single-objective optimization problem in two layer. First, we consider only the performance constraint (C8) and an equality constraint for resource usage translated from (O2) and optimize the average control performance (O1). This is the middle layer of the proposed multi-layer co-optimization. Decision variables related to the objectives, i.e., the sampling periods, are determined. Now, in the inner layer, we synthesize the remaining decision variables satisfying all constraints from (C1) to (C7) based on the results of the middle layer. This process is iterative in the way that if the synthesis fails in the inner layer, we go back to the middle layer for the next best solution. This optimization technique ensures optimality and also efficiency. The success of this nested two-layer approach to solve the single-objective optimization is due to the fact that the feasible region of the design space has good objective values in this problem.

The middle and the inner layers of the co-optimization is given in Algorithm 4. This algorithm takes as input, a discrete value of resource usage U^* and the average control performance J^+ corresponding to the last obtained Pareto point. The outputs of this algorithm are $isPareto$, $\mathcal{P}^*$ and J^*_{av}, where $isPareto$ is a boolean value denoting whether a Pareto point is obtained for the given value of resource usage U^*, and $\mathcal{P}^*$ and J^*_{av} denotes the parameter set and the average control performance corresponding to the Pareto point (if obtained). We initialize $isPareto$ as false. We use a variable J^- as a lower bound on the average control performance for the Pareto point. J^- is initialized as 0. That is, as J^*_{av} will be a positive number, it initially does not have a lower bound. However, when schedule cannot be obtained corresponding to the best average control performance, we update J^- to determine the next best value of the average control performance. Now, in each iteration of the while loop (lines 1-19), the algorithm starts by trying to find the most suitable candidates based on the upper and lower bound of the control performance J^+ and J^- (line 2). All the determined candidates have the same average control performance. One of the candidates can possibly represent a Pareto point. In the function **optimizeCP** in line 2, an MILP model is formulated and solved according to the objective of average control performance. The candidates are each represented by a set of control parameters $\mathcal{P}^c$, where all the candidates found are represented by the set $\{\mathcal{P}^c\}$. If a set of candidates can be found (lines 5-18), the lower bound on the average control performance J^- will be updated to the obtained value of average performance (line 6), in order to exclude already evaluated candidates in the next iteration. Then the algorithm will evaluate them one by one (lines 8-17). For each candidate, the inner layer (line 9) is called to find feasible schedules. This is represented by the function **findSchedules** in line 9. In this function, an ILP model is formulated and solved without objective for a feasible schedule set. If such a set can be found for a candidate, the algorithm considers this candidate a valid Pareto point and stop evaluating further ones (lines 10-13). If a valid schedule set is not obtained then the next solution from the middle layer in the set $\{\mathcal{P}^c\}$ is evaluated in the next loop iteration (lines 14-15). The algorithm also stops once it has rendered all the already generated candidates infeasible and no further candidates can be found (lines 3-4). In the end, if a feasible solution can be found (line 20), then the function returns the corresponding parameter set $\mathcal{P}^*$, average control performance J^* and a true flag $isPareto$ (line 21). Otherwise, it returns a false flag, an empty parameter set and zero average control performance (lines 22-24).

3.6 Experimental Results

In this section, we use a case study to (i) illustrate the flow of the proposed approach and to (ii) show the importance of the Pareto front for the evaluation of trade-offs for distributed control systems design. Furthermore, an analysis is provided to show the scalability of the approach.

3.6.1 A Case Study

In this case study, we use a system motivated by the applications in the automotive domain. Due to the confidentiality issue, it is difficult to find a case study system actually applied in the industry and obtain all the details including the mathematical model of the plants and the task and message models. Therefore, we use a synthetic case study, consisting of a FlexRay based ECU network implementing five control applications typical to the automotive domain.

Plant models: We consider a system consisting of 5 control applications $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5\}$. For each of the control applications, we use a plant model derived from the automotive domain. $\mathcal{C}_1$ to $\mathcal{C}_5$ represent respectively a DC motor speed control (DCM), a car suspension system (CSS), an electronic wedge brake (EWB), and two variants of the cruise control (CC1) and (CC2). The plants used are described as follows:

- DCM is adapted from [202], where the state variables $x = [x_1 \ \ x_2]^T$ represent the rotational speed of the motor shaft and the armature current. The control input u is the motor terminal voltage. This control model can for example be applied to the wheel speed control in a vehicle. The system matrices for this plant are represented as follows:

$$A = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{3.28}$$

  For this plant, we consider that the objective is to have a control cost of an unit step response lower than or equal to 0.7, where the cost function is given by Eq. (3.2). Here, we assume the value of λ as 0.001. The maximum permissible value of the control input is $u^* = 100$.

- CSS, adapted from [161], has the state variables $x = [x_1 \ \ x_2 \ \ x_3 \ \ x_4]^T$, where x_1 and x_2 represent the position and velocity of the car and x_3 and x_4 are the position and velocity of the mass of the suspension system. The control input u is the force applied to the body by the suspension system. The system matrices can be represented as follows:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & -60 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}. \tag{3.29}$$

  For this plant, the control objective is to reject a disturbance within $0.2\,\text{s}$. Here, we assume that a disturbance brings the system to an initial state $x(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, and therefore, the control objective is written as $y(t) \leq 0.01$, $\forall \ t \geq 0.2\,\text{s}$. The control input must not be more than $u^* = 300$.

- EWB is adapted from the self-reinforced brake-by-wire solution developed by Siemens [203]. Modeling of the wedge results in a second-order system. Two state variables $x = [x_1 \ \ x_2]^T$ are the position and the velocity of the braking wedge, respectively. The DC motor model is simplified and the control input u is the force applied by the motor. The plant model is represented as follows:

$$A = \begin{bmatrix} 0 & 1 \\ 8.3951 \times 10^3 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 4.0451 \end{bmatrix}, \quad C = \begin{bmatrix} 7.9920 \times 10^3 & 0 \end{bmatrix}. \tag{3.30}$$

Here, the control objective is to have a unit step response with a settling time lower than or equal to $0.05\,\mathrm{s}$. The input saturation is equal to $u^* = 1$.

- CC1 is taken from [202], neglecting the dynamics of the powertrain and tires. Here, the state variable x represents the speed of the vehicle and the control input u is the force exerted on the vehicle. The plant can be represented as follows:

$$A = -0.05, \quad B = 0.001, \quad C = 1. \tag{3.31}$$

  For this plant, the control objective is to have a unit step response with a settling time lower than or equal to $0.35\,\mathrm{s}$. The control input must always respect the constraint: $u \leq u^* = 20000$.

- CC2 is used in [117] and derived from [204]. The cruise control system regulates the vehicle speed to follow the driver's command. The state space representation of this system can be written as follows:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}. \tag{3.32}$$

  The control objective of CC2 is to guarantee a settling time lower than or equal to $0.6\,\mathrm{s}$ for a unit step input. The control input must be less than or equal to $u^* = 5000$.

Architecture: The platform architecture considered in this case study, as shown in Figure 3.5, consists of three ECUs connected by FlexRay in a bus topology. Table 3.1 shows the task mapping on the ECUs. Table 3.2 shows the assumed values of WCETs for all the tasks. We further assume that the WCET of the communication tasks are equal and given by $\epsilon = 300\,\mu\mathrm{s}$. Furthermore, Table 3.3 shows two sets of bus parameters that we have considered. Set 1 is obtained from a related work [161]. Set 2 is adapted from an industrial application [205]. According to the original parameter values specified in [205], the length of a static slot is not able to accommodate the largest frame in our case study, the payload of which consists of 4 float sensor data of 4 bytes each. Therefore the static slot length is adjusted to $39.875\,\mu\mathrm{s}$ to accommodate this frame. The bus topology is a common one in automotive domain, which can be found in related works [117, 161, 201]. The number of ECUs, the number of control applications and task mapping are chosen for the ease of demonstration. The applicability of the approach, however, is not limited to this setup. In fact, as will be explained in the scalability analysis in Section 3.6.2, the approach can be applied to a number of different setups with varying system sizes and task mappings.

Table 3.1: Task mapping.

ECUs	**Tasks**
E_1	$T_{s,1}$, $T_{c,2}$, $T_{a,3}$, $T_{a,4}$, $T_{c,5}$
E_2	$T_{a,1}$, $T_{s,2}$, $T_{c,3}$, $T_{s,4}$, $T_{s,5}$
E_3	$T_{c,1}$, $T_{a,2}$, $T_{s,3}$, $T_{c,4}$, $T_{a,5}$

Table 3.2: Task WCETs in µs.

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$
$e_{s,i}$	200	400	200	100	300
$e_{c,i}$	300	600	300	150	450
$e_{a,i}$	100	100	100	100	100

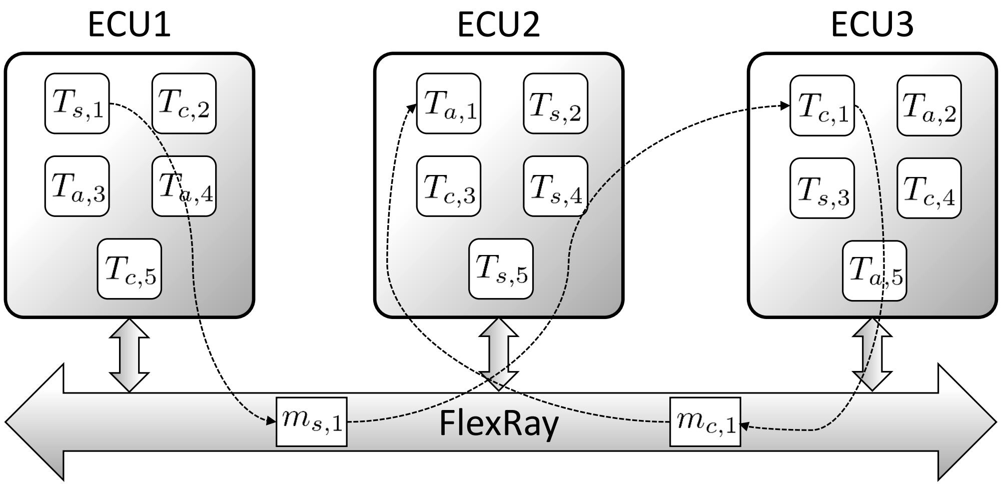


Figure 3.5: The platform architecture in the case study.

Table 3.3: FlexRay Bus Configuration.

Bus Parameters	Values	
	Set 1	Set 2
Bus Speed	10 $Mbps$	10 Mbps
Bus period (T_{bus})	$5\,\mathrm{ms}$	$5\,\mathrm{ms}$
MacroTick	$1\,\mu\mathrm{s}$	$1.375\,\mu\mathrm{s}$
No. of static slots (N_s)	25	75
Number of minislots (N_d)	237	267
Lenth of a static slot (Δ)	$100\,\mu\mathrm{s}$	$39.875\,\mu\mathrm{s}$
Length of a minislot (δ)	$10\,\mu\mathrm{s}$	$6.975\,\mu\mathrm{s}$
No.of configurable communication cycles (N_{com})	64	64
Synchronization constant (γ_{syn})	$50\,\mu\mathrm{s}$	$50\,\mu\mathrm{s}$

Prospective controller design: In this stage, i.e., the first stage of the co-design, we identify prospective controllers at each possible value of the sampling period for a control application. For $T_{bus} = 5\,\mathrm{ms}$ and $N_{com} = 64$, the choices of sampling period are constrained by $h^{(k)} \in \{5, 10, 20, 40, 80, 160, 320\}$ ms. Now, for each application $\mathcal{C}_i$, we take a value for the sampling period $h_i = h^{(k)}$ from the predetermined set and with the closed-loop delay being equal to the sampling period $d_i = h_i = h^{(k)}$, we design an optimal controller using Algorithm 2. This algorithm basically carries out PSO-based search for optimal closed-loop poles.

Figure 3.6 shows the normalized control performance of the optimal controller obtained at each sampling period for the five control applications. Here, we plot the normalized control performance because this allows us to illustrate different performance metrics in a single graph. The absolute control performance can, however, be calculated from this graph and the given performance requirement.

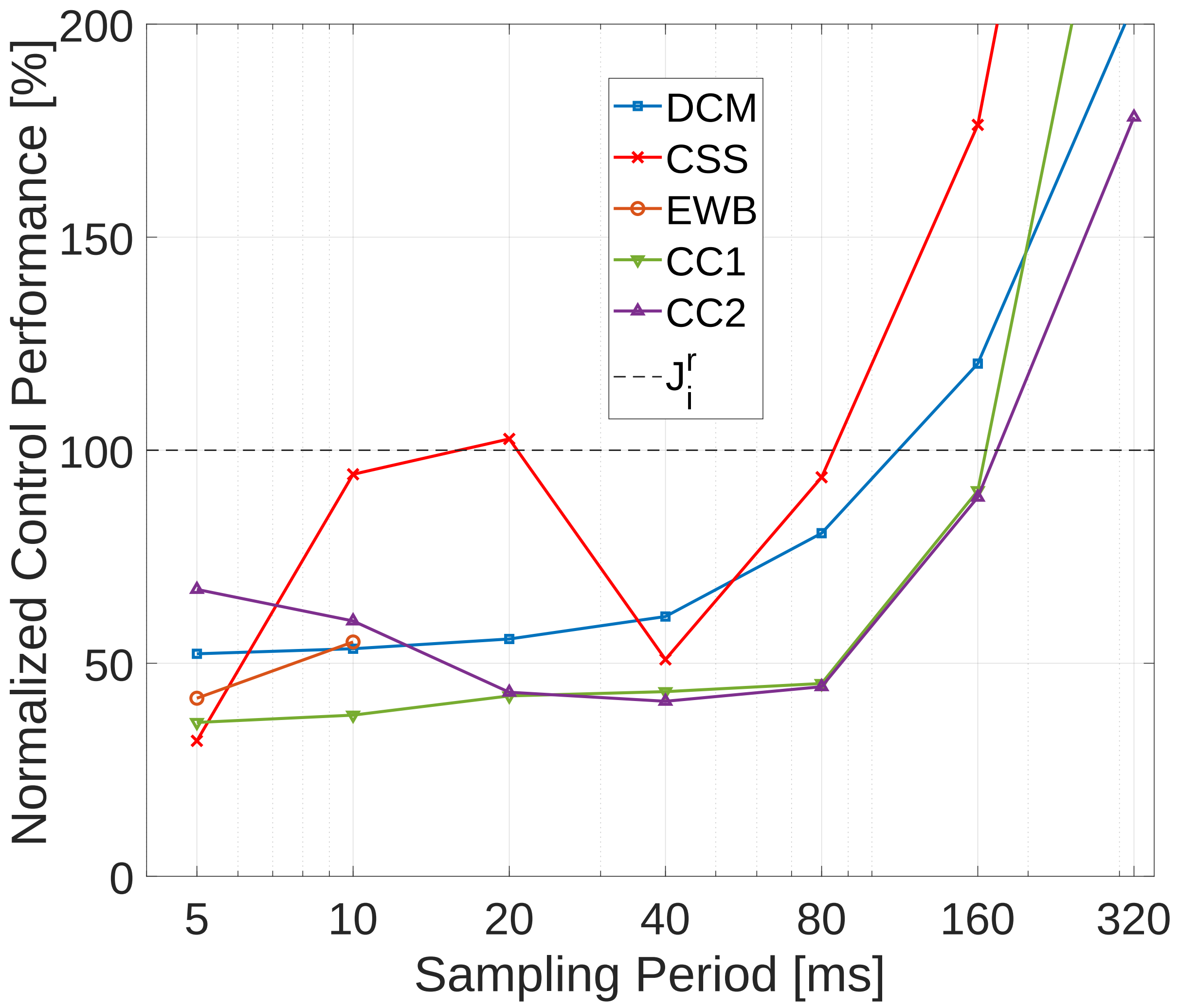


Figure 3.6: Variation of the normalized control performance with feasible values of sampling period for each application. Note that the plot is in semi-log graph where the normalized control performance is plotted in the linear scale while the sampling period is plotted in the logarithmic scale. The black dashed line is drawn horizontally at $100\,\%$ that denote the normalized required performance for all applications. Further note that for EWB, we can design stable controllers only at sampling periods $5\,\mathrm{ms}$ and $10\,\mathrm{ms}$ respectively. For DCM, CSS, and CC1, optimal controllers are respectively obtained also at $320\,\mathrm{ms}$, however, they are not shown for better readability.

For EWB, controllers that stabilize the plant could not be found for five sampling period values, i.e., $20\,\mathrm{ms}$, $40\,\mathrm{ms}$, $80\,\mathrm{ms}$, $160\,\mathrm{ms}$, and $320\,\mathrm{ms}$ respectively. For all other applications, stable controllers are obtained at each of the feasible sampling period values. Note that, here, we do not say that a stable controller does not exist for EWB at the aforementioned sampling period values. However, Algorithm 2 could not find discrete-time poles within a unit circle for which the control input also respects the constraint $u[k] \leq 1$, $\forall k \in \mathbb{Z}^*$. Here, we have considered a timeout of $1000\,\mathrm{s}$, within which all particles must be assigned their respective initial positions via random number generation (lines 3 to 16 in Algorithm 2), otherwise the algorithm returns that the controller design is infeasible for the given plant model, sampling period, delay, and input saturation.

The black dashed line in the plot shows the normalized required performance for all control applications (i.e., 100%). Only the points below or on the black dashed line meet the performance requirements. This enables efficient design space pruning. Here, DCM has five controllers that meet the requirements, while CSS has four, EWB has two, and CC1 and CC2 have six each. However, it can be further observed in Figure 3.6 that for CC2 (i.e., plotted using the purple line marked with triangles pointing upwards), the control performances corresponding to the sampling period values of $5\,\mathrm{ms}$, $10\,\mathrm{ms}$, and $20\,\mathrm{ms}$ respectively are worse compared to the optimal performance obtained for the sampling period of $40\,\mathrm{ms}$. Thus, for CC2, the optimal controllers designed for the sampling period values of $5\,\mathrm{ms}$, $10\,\mathrm{ms}$, and $20\,\mathrm{ms}$, will never be selected by the optimizer in the next stage. This is because there already exists a better controller using less resources, and therefore, implementing a low-performance yet resource-intensive controller is highly inefficient in that case. After design space pruning, there are only three prospective controllers for CC2 that are relevant for the co-optimization stage. Similarly, CSS has three prospective controllers, EWB has two, DCM has five, and CC1 has six.

Although it is expected that the control performance improves with a shorter sampling period, it might be observed in Figure 3.6 that for DCM and CC1, at lower values of sampling period, the control performance does not vary appreciably. The main reason for this smaller variation is the constraint that we have considered on the control input. Typically, with a lower sampling period, higher values of control input can be appropriately managed that enables faster stabilization of the system. However, we have considered that the control input cannot be increased infinitely. At lower sampling periods, control input gets saturated and therefore, cannot improve the performance as expected. Furthermore, for CC2 and CSS, the control performance does not always improve with a lower sampling period. Here, note that with a higher sampling period, the maximum permissible input is applied to the plant for a longer time, and if this does not cause an overshoot, it can stabilize the system faster. Also, it might be that PSO has not been able to find the optimal set of poles by a significant margin in certain cases.

The runtime of Algorithm 2 does not only depend on the number of closed-loop poles to be searched, the number of particles $ParNum$, and the number of iterations $IterNum$. The initialization of the particles in the design space (lines 3 to 16 in Algorithm 2) also depends on the plant model and the maximum permissible value of the control input. For example, if the control input is too constrained, it is challenging to randomly place particles such that they respect the input saturation constraint, i.e, $u[k] \leq u^*$, $\forall k \in \mathbb{Z}^*$. For our experiments, the maximum time this algorithm has taken to run is approximately $10\,\mathrm{min}$ and that is for the fouth-order system CSS and the third-order system CC1, while the minimum time is taken for first order system CC1 which is around $2\,\mathrm{min}$. These times are acceptable considering that the design is carried out offline. Furthermore, for EWB, when the sampling period is $20\,\mathrm{ms}$, $40\,\mathrm{ms}$, $80\,\mathrm{ms}$, $160\,\mathrm{ms}$, and $320\,\mathrm{ms}$ respectively, the algorithm has hit the timeout of $600\,\mathrm{s}$ for the initialization stage of the PSO, and therefore, we could not design stable controllers in these cases. Note that we run the algorithm with 100 particles and up to 100 iterations.

Co-optimization: In the co-optimization stage, we use two different sets of bus parameters as given in Table 3.3. The Pareto fronts obtained in the co-optimization stage for both sets

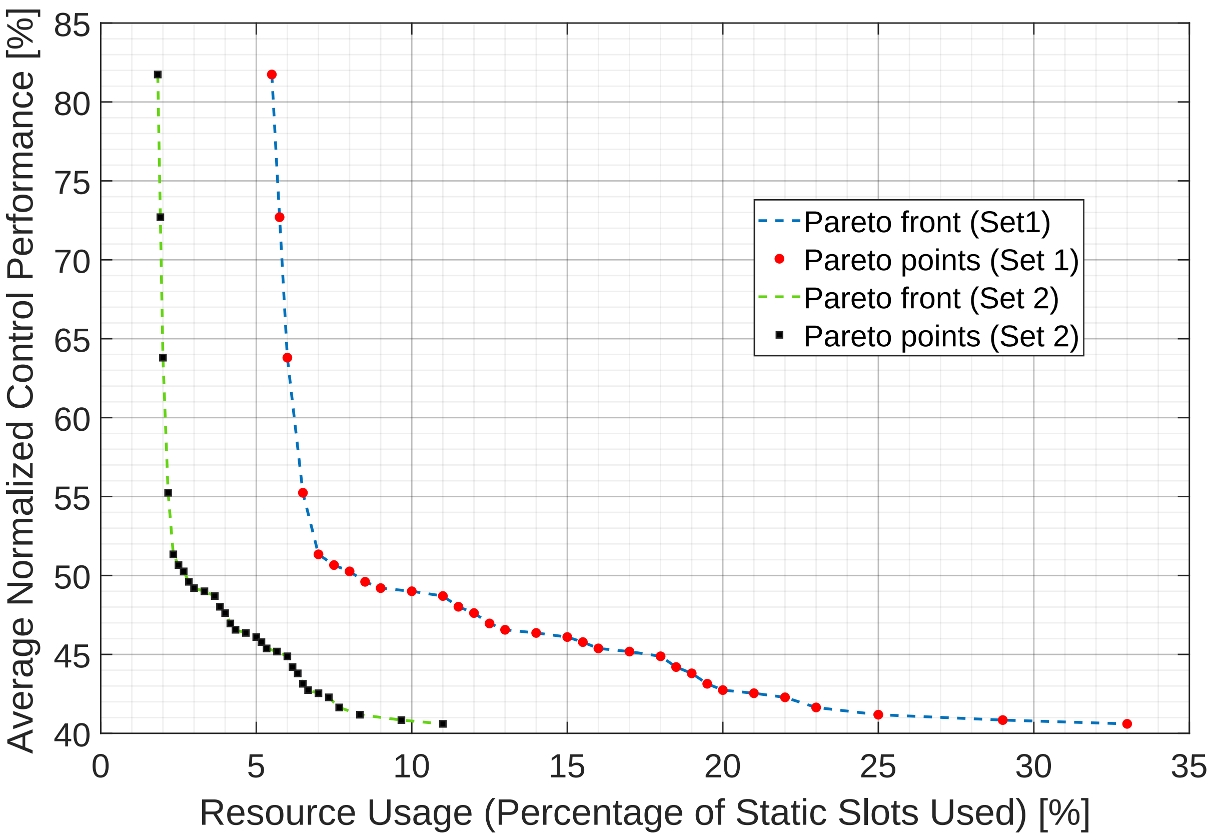


Figure 3.7: Pareto fronts showing the trade-off between average control performance and resource usage for the case study consider two different bus configurations as given in Table 3.3. Note that the resource usage is illustrated, here, as the percentage of static slots being used by the control applications. There are 31 Pareto points in each of the Pareto fronts, where each Pareto point represents a valid Pareto-optimal design configuration for the system under study.

are shown in Figure 3.7. It can be observed in the figure that the proposed approach obtains 31 Pareto points for each bus configuration. Each point represents a design option that offers a specific choice of trade-off between resource usage and average control performance. The developer can then choose one design option according to his/her own preference. For Set 1, (i) the value of the resource usage – percentage of static slots used – ranges from $5.5\,\%$ to $33\,\%$, and (ii) the value of the average control performance varies from $40.6\,\%$ to $81.74\,\%$. It should be noted that for the control performance defined in this paper, the smaller the value, the better the performance. It is obvious that there is a large freedom among these viable designs. If the resource efficiency is the top design priority, the engineer can only use $5.5\,\%$ of the bus bandwidth in the static segment to achieve stable control. If a performance-optimal design is desired, a much better performance (average improvement of $41.14\,\%$) can be achieved at the cost of an extra $27.5\,\%$ of the bandwidth. Therefore, the Pareto points obtained can be explored to obtain a design choice most suitable for the requirement. Note that for a relatively small system size, there is already such a considerable design freedom available. For larger systems, an engineer could possibly profit even more from the trade-off choices between the two objectives.

For Set 2 comprising a different bus configuration, the number of Pareto points is exactly the same as for Set 1, although each Pareto point corresponds to a different set of schedules for the tasks and the FlexRay messages. The difference here is that the number of static slots in one communication cycle is 75 instead of 25. Therefore, the percentage of static slots used in all design options in the Pareto front ranges from $1.83\,\%$ to $11\,\%$ in this case. In terms of control performance, the values are the same as for Set 1.

Furthermore, note that, not for all discrete values of resource usage, a Pareto point exists. The reason for this could be that either (i) for such a resource usage value, a feasible parameter set can not be synthesized, or (ii) the optimal solution for this value is dominated by other points, and therefore, it cannot be considered as a Pareto point.

For the case study, the co-optimization stage for Set 1 and Set 2 took $27\,\mathrm{s}$ each on a laptop with an Intel(R) Core(TM) i7-8550U processor of 1.80GHz and 16GB RAM. The approach is implemented primarily in Matlab while using the Gurobi [172] optimizer in the inner layer and the CPLEX solver [172] in the middle layer of the co-optimization to solve the ILP and the MILP problems respectively. Here, Gurobi is significantly faster than CPLEX in solving the scheduling problem in the inner layer while we exploit the *solution pool* feature of CPLEX in the middle layer to obtain all optimal solutions.

Software simulation using COTS tools: In Chapter 4, we propose a toolchain that integrates the control-platform co-design technique introduced here into COTS tools for automotive software development. Using the proposed toolchain, we synthesize software for the system under study, as will be outlined in Section 4.4. Correspondingly, we demonstrate, using the COTS tools, functionality and schedule co-simulations for the design configurations representing a few Pareto points.

3.6.2 Scalability Analysis

In order to evaluate the scalability of the proposed approach, we have conducted an analysis on a set of synthetic test cases of different system sizes. We consider that the system size range from 6 to 36 control applications. When the number of control applications increases, we also increase the number of ECUs. We define η as the ratio between the number of control applications and the number of ECUs in the whole system and η reflects the average ECU load. We consider three different cases, i.e., $\eta = 1$, $\eta = 1.5$ and $\eta = 2$ respectively. We consider the five plant models, i.e., DCM, CSS, EWB, CC1, and CC2, as given in Section 3.6.1. For each plant model, we choose five different performance requirements, i.e., $\{0.8, 0.9, 1, 1.1, 1.2\} \cdot J_i^r$, where J_i^r is the value of the performance requirement for the case study in Section 3.6.1. Thus, we obtain 25 different look-up tables storing normalized optimal control performances for different choices of sampling period. Now, for each system size and a value of η, 10 test cases are randomly generated including task mapping on ECUs, WCETs of tasks, and a set of look-up tables selected from the predetermined pool. We use a similar FlexRay bus configuration as the Set 1 shown in Table 3.3. The only difference is that the dynamic segment is turned into extra 23 static slots.

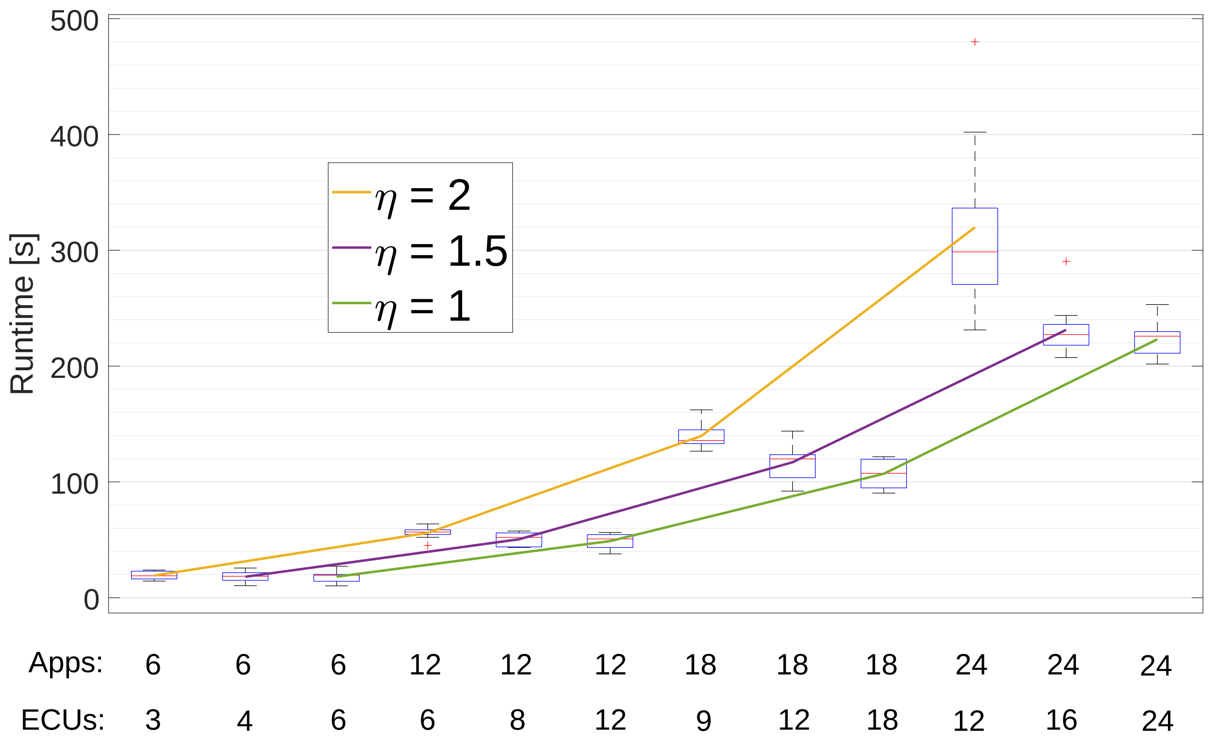


Figure 3.8: Runtime of the co-optimization stage for different problem sizes. In the x-axis, we show the problem size for each box plot that is defined by the number of applications and the number of ECUs in the co-design problem. Here, each box-plot shows the variation in the runtime of the co-optimization algorithm for the given problem size. Furthermore, the increase in the average runtime with the number of applications for three different values of η is also shown, where η is the ratio between the number of applications and the number of ECUs.

Table 3.4: Scalability analysis.

η		No. Control Applications					
		6	12	18	24	30	36
1	Mean runtime	18.18 s	48.91 s	107.04 s	223.11 s	734.5 s	4887 s
	Mean no. of Pareto points	36.6	75.4	124.4	162.7	203.5	248.1
1.5	Mean runtime	18.1 s	50.45 s	116.99 s	231.29 s	804 s	3474 s
	Mean no. of Pareto points	34.5	74	119.2	164.1	214.2	245.5
2	Mean runtime	19.16 s	55.98 s	139.69 s	319.86 s	1011 s	6477 s
	Mean no. of Pareto points	34.2	75.4	119.4	165.4	204.8	245.6

For better visualization, the times for executing the co-optimization stage in 120 test cases are depicted as box plots corresponding to different system sizes up to 24 applications, as shown in Figure 3.8. Futhermore, for each system size and a value of η, the average runtime of the co-optimization stage and the average number of Pareto points obtained are reported in Table 3.4. It can be observed that with increase in the number of applications to be implemented, there is a significant (approximately exponential) increase in the runtime of the co-optimization stage. This is also expected because we solve an MILP and an ILP in the middle and the inner optimization layers respectively and such optimizations can only be solved with exponential complexity. Note that with increase in the number of applications, the number of tasks and messages increases, and therefore, there is a linear increase in the number of unknowns in the optimization problem, thereby increasing the runtime exponentially.

Nevertheless, our main goal in this work is to solve the control-platform co-design problem for control applications within a bus cluster. From the obtained results, we can say that our proposed approach can easily scale to a system size that approximately represents a bus cluster. We study two real-world automotive benchmark subsystems: (i) In [206], the system comprises 9 ECUs, 44 tasks and 19 messages. (ii) In [207], the benchmark system contains 15 ECUs and 53 messages. For 30 applications, we have 90 tasks and 60 messages in the system, which is larger than both benchmarks. In [206], the task to ECU ratio is 4.9 and the message to ECU is 2.1, and in [207], the message to ECU ratio is 3.5. Therefore, we consider the value of η up to 2, which results in the task to ECU ratio of 6 and the message to ECU ratio of 4. For 30 applications and $\eta = 2$, the average runtime of the co-optimization stage is $1011\,\mathrm{s}$, while the maximum time taken is $1294\,\mathrm{s}$. This is very reasonable for an offline process. On average, 204 Pareto points are generated for 30 applications mapped on to 15 ECUs. Note that we have used a time limit of 1 minute and 2 minutes to solve the ILP and the MILP problems in the middle and the inner layers respectively. Our analysis shows that in certain test cases, one or more Pareto points correspond to up to 80% usage of the FlexRay static segment and up to 55% average ECU utilization.

3.7 Related Works

In this chapter, we have outlined our work towards multi-objective co-optimization in control-platform co-design for FlexRay-based distributed control systems. In the context of this chapter, the related works can be organized into four categories as follows:

- In this work, we have considered scheduling the control-related messages only in the static segment of a FlexRay bus cycle. Optimal scheduling in the FlexRay static segment has been studied in the past. [208] has proposed a genetic algorithm based schedule optimization for the FlexRay static segment. [209] has further proposed an integer linear programming formulation for a customized software architecture. [210] has considered slot multiplexing while scheduling the FlexRay messages and has proposed simple heuristics for schedule optimization. [200] has studied the AUTOSAR specifications, considering which a two-dimensional bin packing problem is formulated for scheduling messages in the FlexRay static segment. An ILP based optimal solution and an efficient heuristic is proposed. In these related works, scheduling FlexRay messages are only considered, however, in our work, we considering co-scheduling the tasks and messages together with the design of controllers for distributed automotive applications.

- The task and network schedule co-synthesis problem has been studied in the literature for different system settings. In the context of time-triggered systems, [100, 101] have studied the co-synthesis problem considering communication over a TTEthernet communication network, while [211, 212] have considered FlexRay-based distributed systems. While [100] has formulated the problem as an SMT model, [101, 211, 212] have provided an LP formulation. Note that our approach of co-scheduling the tasks and messages in the inner layer of the co-optimization is similar to these related works. However, these existing works do not consider the interplay between the task and message schedules and the control parameters, which is the main focus of our work.

- In the context of control-platform co-design, [149] has proposed a method that integrates the controller design and the task and message scheduling, and optimizes the overall control performance. In [149], genetic algorithm is employed to search for the optimal assignment of sampling periods to the control applications. Later, [161] has proposed a constraint-driven synthesis for the co-design of controllers with task and FlexRay schedules. A stable controller is designed first for each control application assuming a permissible sampling period, and based on the pre-selected sampling periods, the schedules are synthesized for the applications. [201] has proposed a co-design problem formulation for FlexRay-based systems where the design of the controllers and the schedules are integrated into a holistic framework with control performance as the only optimization objective. This work considers both variable sampling period and delay in the controller design. Note that [149, 161, 201] have not considered the trade-off between multiple optimization objectives. Furthermore, some approaches appear difficult to scale. For example, in [201], it already takes more than one hour to synthesize a system of 5 applications, among which only 3 are control applications.

- Multi-objective optimization has been applied in scheduling of wireless communication networks [213,214]. Both time and energy consumption are optimized in [213], when scheduling a wireless sensor network. In [214], radio resource scheduling is considered, aiming to optimize outage, capacity and throughput, for the given available resources. The significance of applying multi-objective optimization techniques in the problems naturally embedded with more than one design objective is that the trade-off among various objectives can be analyzed and that the decision maker is given the flexibility to choose a design point based on customized situations. But these related works do not address the specific problem in the control-platform co-design domain.

3.8 Conclusion

In this chapter, we have proposed a two-stage approach for the design of distributed control systems in an automotive setting. In the first stage, for each application, we design prospective optimal controllers at different sampling periods as allowed by the underlying implementation platform. Here, we use PSO-based optimal controller design technique to synthesize prospective controllers. Using the results from the first stage, we formulate a co-optimization problem in the second stage with the control and platform parameters as the problem variables, while the resource usage and the average control performance are considered as the optimization objectives. To solve the co-optimization problem, we propose a multi-layer hybrid optimization approach in the second stage exploiting the problem characteristics. In the outer layer, we iterate through all possible discrete values of resource usage. For a given value of resource usage, in the middle layer, we determine the sampling periods of the applications that optimize the average control performance by solving an MILP problem. In the inner layer, for given sampling periods, we determine a feasible set of task and message schedules by solving an ILP problem. The proposed co-optimization approach generates a Pareto front to offer design trade-offs between the objectives of average control performance and resource usage.

In the future, we would like to consider a more sophisticated controller design technique considering variable sensor-to-actuator delays, while not making a compromise on the scalability of the approach. Secondly, we have considered here only the co-optimization problem within a single FlexRay bus cluster. Future works may extend this to multiple FlexRay bus clusters, or even to heterogeneous networks comprising bus clusters exhibiting different communication protocols. The challenges here include firstly the scalability to even larger system sizes and possible trade-offs between more optimal designs and less computational effort. In addition, to deal with less deterministic bus systems, extensions like timing analysis might be required. Thirdly, in this work, we assume that each message is sent as a FlexRay frame. However, to better utilize the static slots in FlexRay, we can also consider to pack several messages into one FlexRay frame. This will increase the design dimensions, and hence, the complexity of the co-design problem. Thus, there are still several opportunities to improve the resource usage and/or control performance in the design of distributed control systems, however, the challenge is to handle the design complexity for a more holistic consideration.

4
Tool Integration for Automated Synthesis and Implementation of Distributed Cyber-Physical Systems[1]

4.1 Introduction

In practice, the controller design and the software implementation for distributed embedded controllers are carried out in their isolated design spaces using respective COTS design tools. Nevertheless, the state-of-the-art along control-platform co-design made some progress over the last few years, where the emphasis has been on joint co-synthesis for system-level optimization. While the general challenge for these approaches is scalability, the lack of integrated industrial tools has prevented these techniques to be evaluated in industrial settings. In effect, this leads to a gap between the state of the art and the state of practice. In Chapter 3, we have proposed a scalable control-platform co-design technique. Now, in this chapter, we introduce a toolchain that integrates the proposed co-design approach into commercially available design tools to bridge the gap between co-design schemes and the implementation of control software.

Conventional design and implementation flow: We study the implementation of controllers on an automotive embedded platform comprising ECUs connected over a FlexRay bus. Furthermore, we review a toolchain, commonly found in the automotive domain, that enables design and development of FlexRay-based systems. The toolchain consists of MATLAB/Simulink for controller design and SIMTOOLS/SIMTARGET toolboxes [216, 217] for platform configuration and software implementation. Correspondingly, we have studied the conventional design

[1]This chapter is based on a publication entitled "Tool integration for automated synthesis of distributed embedded controllers" [215] that will appear in the ACM Transactions on Cyber-Physical Systems (TCPS).

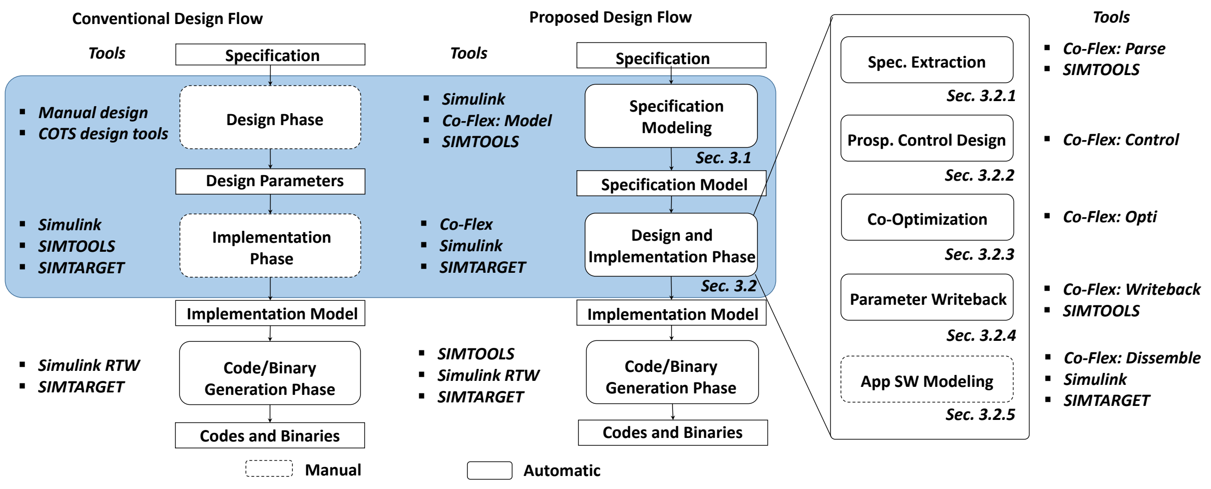


Figure 4.1: Schematic of the proposed framework and the toolchain support

and implementation flow of automotive embedded controllers using such a toolchain [218] as shown in Figure 4.1.

The development of automotive control software can be partitioned into three subsequent phases, namely (i) the *design phase* – that calculates and validates the parameters like the control gains and the task and communication schedules from the specification (including, e.g., control plant models, performance criteria and architecture model), (ii) the *implementation phase* – where the system and application software is modeled and configured using the parameters obtained from the design phase, (iii) the *code generation phase* – where the implemented models are used to generate code and binary files for deployment on the hardware.

The available toolchain automates certain parts of the development as follows: (i) SIMTOOLS/SIMTARGET provide specific blocksets that enable modeling of FlexRay network and ECU, partitioning and mapping of tasks, packing of messages into frames, configuration of task and message schedules, and definition of input and output interfaces. (ii) The Simulink Realtime Workshop along with SIMTARGET can be used to generate C-code and binary files.

However, the design flow with the aforementioned toolchain also involves the following manual processes and is thus tedious, time-consuming and error-prone. (i) In the design phase, specification needs to be correctly interpreted to manually formulate the parameter synthesis problem that can then be solved either manually or using some COTS tools. For example, the control gains can be designed based on MATLAB/Simulink model of the plant by closed-loop simulation of the plant and the controller. Subsequently, the control gains can be manually tuned, and the ones corresponding to which closed-loop system meets higher level performance requirements, are chosen as design parameters. On the other hand, the schedule synthesis problem can be formulated manually as a constraint programming problem [211] considering schedulability constraints, data dependencies and assumptions made on the control side on the values of sampling period and delay. Subsequently, the problem is solved using some commercial solvers like Gurobi and Z3. However, when no feasible schedule set exists, the design steps are reiterated. It may be noted here that for such an iterative design paradigm it is very time-consuming to explore the trade-off between different objectives. (ii) In the implementation phase, high level design tool, i.e., Simulink is used to manually model the application and system software, with the support of SIMTOOLS/SIMTARGET for FlexRay related and Electrobit hardware dependent software design. The software model must be manually configured with the calculated values of control and platform parameters.

Proposed design and implementation flow: Towards the aforementioned shortcomings of the conventional flow, we propose a MATLAB/Simuink and SIMTOOLS/SIMTARGET based toolbox, **Co-Flex**, that assists the designer by bridging the gaps between the aforementioned COTS tools, automating most of the manual processes discussed above and offering the engineer a larger design freedom. Towards reducing manual intervention, Co-Flex offers (i) template blocks that can be conveniently used to model automotive control applications through easy parametrization, and (ii) specific functionalities that automate the flow between different design phases, e.g., specification extraction from template models, re-configuration of the control models with the obtained values of control parameters, and synthesizing the implementation model with correct platform parameters, i.e., task and message schedules. In addition, Co-Flex also introduces the novel co-optimization approach (as discussed in Chapter 3) for simultaneously synthesizing the control and platform parameters considering different trade-offs between the average control performance and resource usage, thereby offering more design choices.

With Co-Flex, the first two phases in the conventional design flow can be replaced by a *specification modeling* phase and a *design and implementation phase*, as shown in Figure 4.1. This is done to considerably reduce the manual efforts in the design and the implementation phases of the conventional flow. In the specification modeling phase, **Co-Flex: Model** blocksets can be used together with Simulink/SIMTOOLS/SIMTARGET to develop a template software model that is configured according to the design specifications, i.e., control plant models, architecture model, performance requirements, among others. Subsequently, the design and implementation phase is composed of five stages as shown in Figure 4.1.

In the first stage, i.e., the *Specification Extraction*, **Co-Flex: Parse** tool can be used to automatically extract the specification from the template model. Stage 2 and 3, namely the *Prospective Controller Design* and the *Co-optimization* respectively, implement the co-design approach proposed in Chapter 3. In the prospective controller design stage, **Co-Flex: Control** tool is invoked for each application that synthesizes an optimal controller at each possible sampling period. This is done by using the pole placement controller design method and exploring the design space using the particle swarm optimization (PSO) technique, as given in Algorithm 2. By designing first the prospective controllers, we avoid unnecessary schedule synthesis for sub-optimal or unstable controllers. Subsequently, in the co-optimization stage, **Co-Flex: Opti** tool formulates a bi-objective optimization problem according to the specification extracted and employs the multi-layer hybrid optimization technique, as described in Section 3.5, to generate a number of feasible design parameter sets, where each set represents a Pareto point reflecting the trade-off between the objectives of average control performance and resource usage respectively. The designer can then select a parameter set corresponding to a Pareto point in the Pareto front according to the design requirements. Based on the designer choice of the Pareto point to be implemented, in the *Parameter Writeback* stage, **Co-Flex: Writeback** tool can automatically interpret the synthesis result obtained from the prospective control design and the co-optimization stage and configure the software model with the appropriate values of control and platform parameters. Finally, in the *Application Software Modeling* stage, the **Co-Flex: Dissemble** tool gets rid of the specification models that were required only for the design and need not be a part of the implementation. In addition, the designer can manually add application-specific details to the model if required.

Contributions: Extending the novel control-platform co-design technique proposed in Chapter 3, in this chapter, we make the following contributions:

- We introduce a novel design and implementation flow for FlexRay-based distributed control systems that enables the application of a state-of-the-art co-design scheme and makes the whole process much more convenient. Compared to the conventional approach where the design and implementation is carried out successively, in the proposed development process, we first create a partial model of the system based on the specification. Using this partial system model, we synthesize the design parameters that are then used to model the remaining parts of the system. ECUs can then be directly flashed using the codes generated from the developed model.

- We develop a toolchain support that can automate the design process of distributed controllers based on the introduced design and implementation flow. This toolchain enables automated modeling of distributed control systems through easy parametrization. It further enables the simultaneous design of control and platform parameters while co-optimizing the average control performance and the resource usage respectively by incorporating a tool that implements the proposed co-design approach as described in Chapter 3. The proposed toolchain is compatible with commonly used industrial-strength development tools, i.e., MATLAB/Simulink and SIMTOOLS/SIMTARGET (Elektrobit tools for platform configuration). It integrates the aforementioned tools to enable automated synthesis of control software for FlexRay-based systems.

- We consider a case study comprising five control applications mapped on to three different ECUs communicating over a FlexRay bus. For the case study, we follow the development process and the integrated toolchain proposed here, to generate codes that will run on the ECUs. The model-in-the-loop simulation that is offered by the SIMTOOLS validates the control and platform parameters synthesized using the proposed co-design scheme. Furthermore, we build a realistic distributed setup comprising three Elektrobit ECUs connected by cables (unshielded twisted pair) and D-SUB9 connectors. We flash the software binaries on the three ECUs respectively without any error which implies that the configuration of design parameters have been correct.

Chapter organization: The rest of this chapter is organized as follows. In Section 4.2, we discuss the conventional process for software development of FlexRay-based distributed control systems where MATLAB/Simulink is used for controller modeling and design while platform modeling and configuration is supported by Elektrobit tools, i.e., SIMTOOLS/SIMTARGET. We further point out the shortcomings of the conventional development process. Thereafter, in Section 4.3, we describe the proposed integrated toolchain for the design and implementation of FlexRay-based distributed control systems, while we also outline the step-by-step process for the automated software development of such systems using the integrated toolchain. In

Section 4.4, we show a case study comprising five control applications implemented on three ECUs connected by a FlexRay bus, for which we develop the ECU software using the proposed integrated toolchain while following the suggested development process. The related works are discussed in Section 4.5 and we provide the concluding remarks in Section 4.6.

4.2 Conventional Design and Implementation of Distributed Automotive CPSs

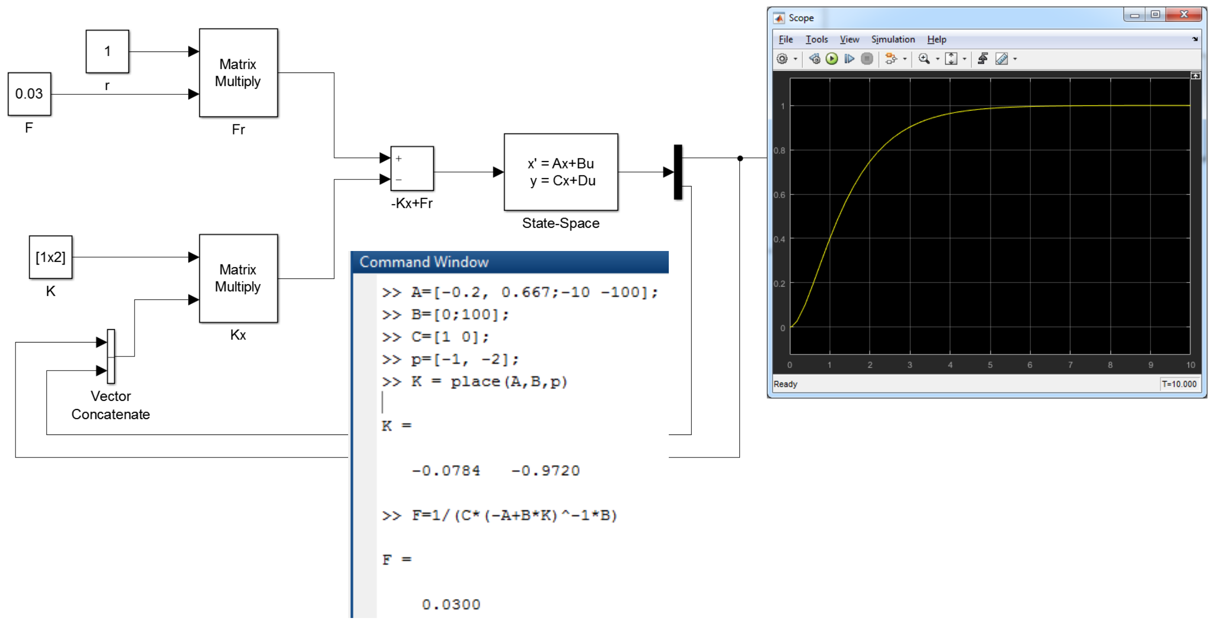
We study the same problem setting as in Chapter 3, where multiple control applications are implemented using a number of ECUs communicating over a FlexRay. The control and platform models for such a setting are provided in Section 3.2.1. Typically, for this problem setting, the conventional design only synthesizes the control and platform parameters while respecting the system constraints, e.g., performance requirements and schedulability constraints, however, without considering any optimization objective. For a given system with a set of control applications, $\mathcal{C}$, the parameter synthesis problem essentially boils down to finding for each control application C_i, (i) the control parameters including the control gains and the sampling period (par_i^c), and (ii) the platform parameters including the task and message schedules (par_i^s).

In order to develop the software for a FlexRay-based ECU network running a set of control applications, the systems and the control engineers usually start with a system specification. On the embedded platform side, the specification typically include (i) ECUs and their hardware and OS characteristics and (ii) the basic parameters of the FlexRay bus, e.g., the length of a communication cycle (T_{bus}), the length (Δ) of a static slot and the number (N_s) of static slots, and (iii) the task partitions ($T_{s,i}$, $T_{c,i}$, and $T_{a,i}$), and the task mapping $\mathcal{T}$. On the control side, the plant models ($\{A_i, B_i, C_i\}$s) and the performance requirements (J_i^rs) need to be specified. Based on the specification, it is possible to develop the control software in three subsequent phases as follows: (i) the *design phase*, (ii) the *implementation phase*, and (iii) the *code generation and hardware implementation phase*.

4.2.1 Design Phase

In this phase, the system parameters are synthesized and validated based on some theoretical model of the underlying system. As a first step, the control and the embedded systems engineers negotiate and agree on some constraints on fundamental parameters like the sampling periods and sensing-to-actuation delays of the controllers depending on the platform architecture. For example, ECUs running OSEK/VDX operating systems offer only a predefined set of sampling periods. Similarly, the sensing-to-actuation delay of a controller is constrained by non-negligible time taken by the tasks running on the ECUs and the data transmitted over the communication bus. Execution time of a task on an ECU depends on the processor speed, the memory architecture, and the scheduling scheme run by the OS. The transmission time of a message over the communication bus depends on the communication protocol and the bus bandwidth.

Now, the control engineer tries to calculate the control gains, sampling period, and the closed-loop delay for each of the applications separately based on the respective plant models



```
>> A=[-0.2, 0.667;-10 -100];
>> B=[0;100];
>> C=[1 0];
>> p=[-1, -2];
>> K = place(A,B,p)

K =

   -0.0784   -0.9720

>> F=1/(C*(-A+B*K)^-1*B)

F =

    0.0300
```

Figure 4.2: A controller is designed using MATLAB/Simulink. Control gains are calculated using standard MATLAB functions while the controller and the plant models are connected in closed-loop and are simulated to evaluate the control response.

and satisfying the preliminary constraints on sampling periods and sensing-to-actuation delays. Typically, the control engineer uses modeling tools like Simulink to develop the plant and the controller models for an application. Then, the control gains, the sampling period and the sensor-to-actuator delay for the controller are tuned using closed-loop simulations of the controller and the plant. The parameter values corresponding to which the closed-loop system satisfies the performance requirement are chosen as design configuration. The process of controller design is shown in Figure 4.2.

After the controller design, embedded systems engineer partitions the software model of each controller into several tasks and maps those tasks onto ECUs depending on the layout of the physical system, e.g., the placement of the sensors and actuators. Now, based on task partitioning and mapping of all the applications, the embedded system engineer tries to generate schedules for tasks and the associated messages between communicating tasks. This schedule synthesis problem can be formulated as a constraint programming problem while considering constraints on designed values of sampling periods and delays, data dependencies, non-overlapping tasks and messages, and other architectural constraints. This problem, thus formulated, can be solved using some commercial solvers like Gurobi, CPLEX, and Z3.

When no feasible schedule exists considering the scheduling constraints, the embedded systems engineer may inform the control engineer to re-design the controllers with updated constraints on sampling periods and delays. As a result, this conventional design paradigm can be iterative and time-consuming as shown in Figure 4.3. Moreover, in order to save time, if the engineers on either side make conservative assumptions then the design becomes resource-inefficient which may not be sustainable in the cost-sensitive automotive domain. Furthermore,

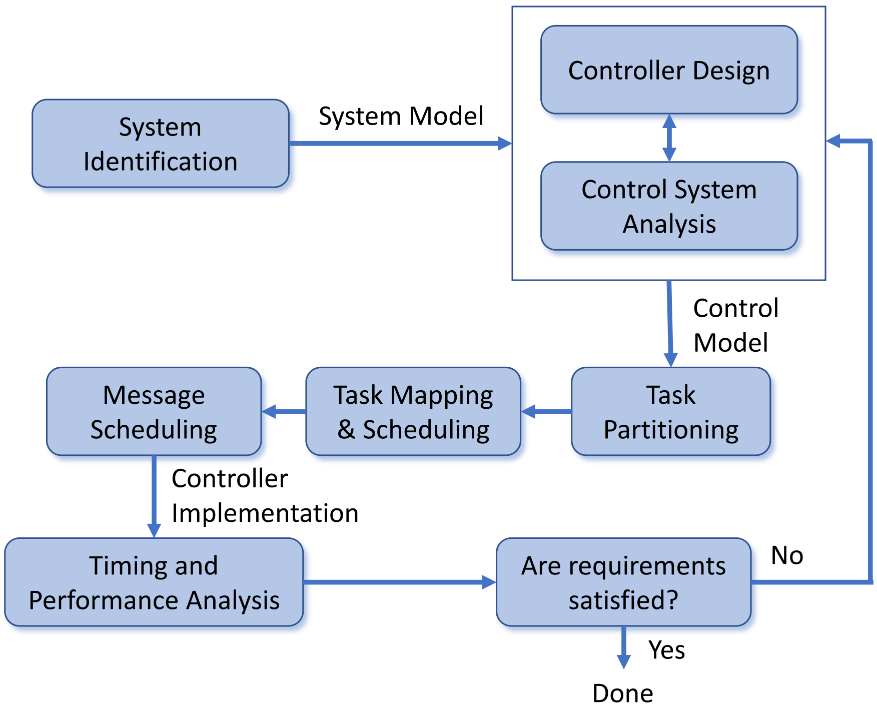


Figure 4.3: Iterative design of control and platform parameters. In this example, control and platform parameters are synthesized consecutively without a proper mathematical interface. The design is followed by analysis of platform timings and control performance. If the analysis suggests that the requirements might not be met, then a redesign might be necessary without any feedback on how to change the design.

this conventional design involves significant manual intervention, and therefore, can be error-prone and tedious.

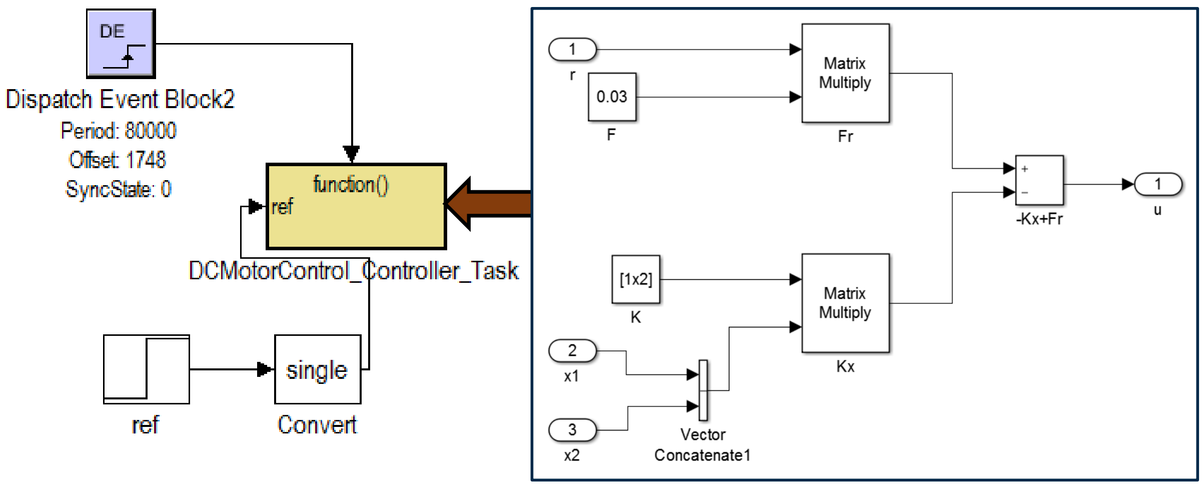


Figure 4.4: Adding the controller model into a task in the implementation phase

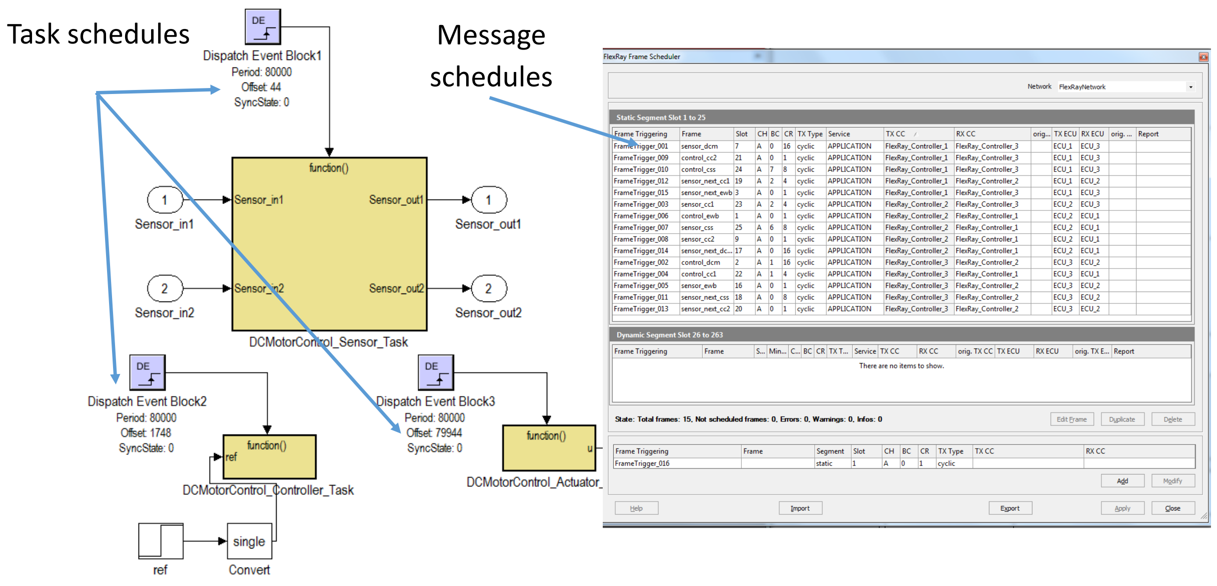
4.2.2 Implementation Phase

In this phase, the whole system is modeled using certain COTS tools. In this work, we study the toolboxes SIMTOOLS and SIMTARGET that can be used in combination with Simulink to develop the system model. The Simulink models of the controllers as developed by the control engineer in the design phase can be reused in this phase. The controller models are combined to form a single model representing the whole system.

Keeping the control laws intact, this model is further manually modified to incorporate the architectural details, i.e., ECUs and the FlexRay bus. SIMTOOLS provides specific blocks to model the ECUs and the FlexRay bus. Details can be added as parameters to these blocks. For example, the type of OS that will run on the ECUs can be selected and different bus parameters like cycle time, static slot length, and number of static slots can be configured.

Furthermore, implementation-specific details like task partitioning and mapping, data mapping and frame packing can be modeled using SIMTOOLS. Model of a task must be inserted into a specific block called ‘Function_Call_Subsystem’ as shown in Figure 4.4. On the other hand, a task can be assigned to a specific ECUs using a ‘Function to ECU and MCU Assignment Block’. Data can be mapped to a FlexRay signal using ‘RTE Signal Communication Block’ and several signals can be packed into a FlexRay frame in the ‘SIM DB’ block.

Next, application-level details can be added to the model, i.e., the details of the sensors and the actuators. Sensor tasks are modeled manually based on the type of sensors and how they are connected. For example, when an absolute encoder is used to sense the speed of a motor, we need to calculate the rate at which the absolute angular position changes, and therefore, in the sensor task, we need to add a derivative block to which the sensor input is connected. Similarly, an actuator task is modeled according to the actuator used and its operating limits. That is, a saturation block must be required to limit the output of the actuator task.



Network: FlexRayNetwork

Static Segment Slot 1 to 25

Frame Triggering	Frame	Slot	CH	BC	CR	TX Type	Service	TX CC	RX CC	orig...	TX ECU	RX ECU	orig. ...	Report
FrameTrigger_001	sensor_dcm	7	A	0	16	cyclic	APPLICATION	FlexRay_Controller_1	FlexRay_Controller_3		ECU_1	ECU_3		
FrameTrigger_009	control_cc2	21	A	0	1	cyclic	APPLICATION	FlexRay_Controller_1	FlexRay_Controller_3		ECU_1	ECU_3		
FrameTrigger_010	control_css	24	A	7	8	cyclic	APPLICATION	FlexRay_Controller_1	FlexRay_Controller_3		ECU_1	ECU_3		
FrameTrigger_012	sensor_next_cc1	19	A	2	4	cyclic	APPLICATION	FlexRay_Controller_1	FlexRay_Controller_2		ECU_1	ECU_2		
FrameTrigger_015	sensor_next_ewb	3	A	0	1	cyclic	APPLICATION	FlexRay_Controller_1	FlexRay_Controller_3		ECU_1	ECU_3		
FrameTrigger_003	sensor_cc1	23	A	2	4	cyclic	APPLICATION	FlexRay_Controller_2	FlexRay_Controller_3		ECU_2	ECU_3		
FrameTrigger_006	control_ewb	1	A	0	1	cyclic	APPLICATION	FlexRay_Controller_2	FlexRay_Controller_1		ECU_2	ECU_1		
FrameTrigger_007	sensor_css	25	A	6	8	cyclic	APPLICATION	FlexRay_Controller_2	FlexRay_Controller_1		ECU_2	ECU_1		
FrameTrigger_008	sensor_cc2	9	A	0	1	cyclic	APPLICATION	FlexRay_Controller_2	FlexRay_Controller_1		ECU_2	ECU_1		
FrameTrigger_014	sensor_next_dc...	17	A	0	16	cyclic	APPLICATION	FlexRay_Controller_2	FlexRay_Controller_1		ECU_2	ECU_1		
FrameTrigger_002	control_dcm	2	A	1	16	cyclic	APPLICATION	FlexRay_Controller_3	FlexRay_Controller_2		ECU_3	ECU_2		
FrameTrigger_004	control_cc1	22	A	1	4	cyclic	APPLICATION	FlexRay_Controller_3	FlexRay_Controller_1		ECU_3	ECU_1		
FrameTrigger_005	sensor_ewb	16	A	0	1	cyclic	APPLICATION	FlexRay_Controller_3	FlexRay_Controller_2		ECU_3	ECU_2		
FrameTrigger_011	sensor_next_css	18	A	0	8	cyclic	APPLICATION	FlexRay_Controller_3	FlexRay_Controller_2		ECU_3	ECU_2		
FrameTrigger_013	sensor_next_cc2	20	A	0	1	cyclic	APPLICATION	FlexRay_Controller_3	FlexRay_Controller_2		ECU_3	ECU_2		

Dynamic Segment Slot 26 to 263

Frame Triggering	Frame	S...	Min...	C...	BC	CR	TX T...	Service	TX CC	RX CC	orig. TX CC	TX ECU	RX ECU	orig. TX E...	Report

There are no items to show.

State: Total frames: 15, Not scheduled frames: 0, Errors: 0, Warnings: 0, Infos: 0

Frame Triggering	Frame	Segment	Slot	CH	BC	CR	TX Type	TX CC	RX CC
FrameTrigger_016		static	1	A	0	1	cyclic		

Figure 4.5: Manually configuring the task and message schedules in SIMTOOLS blocks.

The updated system model is then manually configured according to the task and message schedules obtained from the design phase. SIMTOOLS provides a ‘Dispatch Event’ block that can be used to specify the schedule of a task as shown in Figure 4.5. While the schedules of FlexRay frames can be added in the ‘SIM DB’ block, as illustrated in Figure 4.5. After the complete modeling, typically simulations are run to validate the correctness of the implemented system. For example, SIMTOOLS offers different simulation option to validate both functionality and timing correctness.

4.2.3 Code Generation and Hardware Implementation Phase

In this phase, the binary file for each ECU is generated from the implemented system model and then flashed. Firstly, the complete software implementation needs to be split into separate model for each of the ECUs. Next, C-code and subsequently binary files are generated for each of the ECUs from the corresponding model. Here, SIMTOOLS offers a function named *Split and Build* that automates the splitting of the system model into separate models for the ECUs. Subsequently, it generates C-code and binary files for each ECU by invoking Simulink Real-Time Workshop (RTW) together with SIMTARGET. Simulink RTW facilitates code generation for Simulink blocks while SIMTARGET generates codes from SIMTOOLS/SIMTARGET blocks. The generated binary files are used to flash the corresponding ECUs.

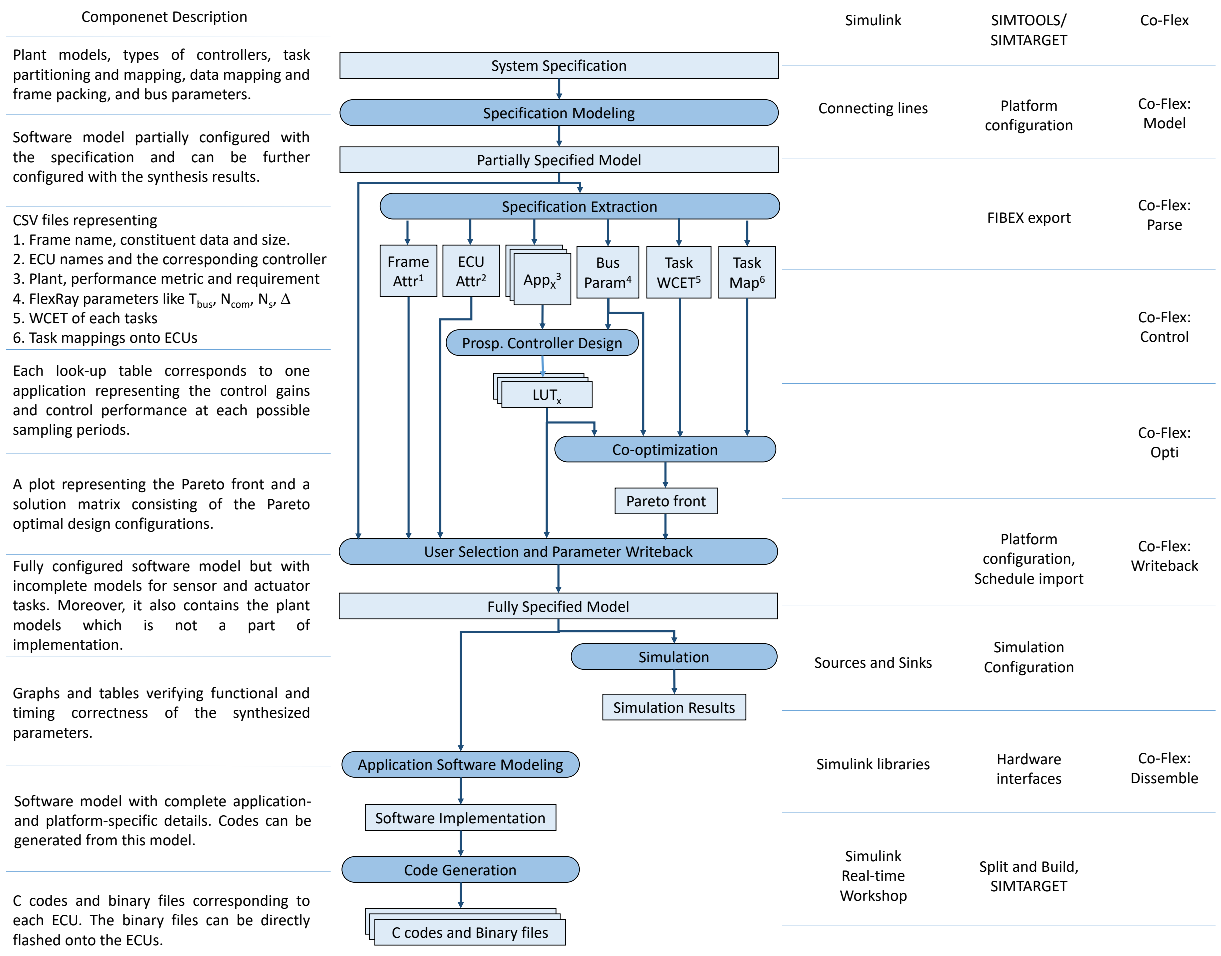
4.2.4 Limitations of the Conventional Approach

The conventional design and implementation of distributed automotive CPSs has the following limitations:

- The developer needs to model the system from scratch which is time-consuming.
- The developer needs to manually formulate the control and platform design problems from the specification. Moreover, in the software implementation phase, the model is manually configured with the designed control and platform parameter values. This is error-prone, time-consuming and cumbersome.
- The final implementation is not efficient both in terms of control performance and resource usage.

4.3 Automated Software Synthesis using the Proposed Integrated Toolchain

In our work, we propose a new design and implementation flow for distributed control systems, as shown in Figure 4.6. We have also applied the proposed flow for the development of FlexRay-based automotive control software. Aligned to the proposed flow, we have further developed a toolchain, named *Co-Flex*, to support such software development in MATLAB/Simulink environment in conjunction with SIMTOOLS/SIMTARGET toolboxes. The proposed flow along with Co-Flex can overcome the disadvantages of the conventional flow in the following ways.


Componenet Description

Plant models, types of controllers, task partitioning and mapping, data mapping and frame packing, and bus parameters.

Software model partially configured with the specification and can be further configured with the synthesis results.

CSV files representing
1. Frame name, constituent data and size.
2. ECU names and the corresponding controller
3. Plant, performance metric and requirement
4. FlexRay parameters like T_{bus}, N_{com}, N_s, Δ
5. WCET of each tasks
6. Task mappings onto ECUs

Each look-up table corresponds to one application representing the control gains and control performance at each possible sampling periods.

A plot representing the Pareto front and a solution matrix consisting of the Pareto optimal design configurations.

Fully configured software model but with incomplete models for sensor and actuator tasks. Moreover, it also contains the plant models which is not a part of implementation.

Graphs and tables verifying functional and timing correctness of the synthesized parameters.

Software model with complete application- and platform-specific details. Codes can be generated from this model.

C codes and binary files corresponding to each ECU. The binary files can be directly flashed onto the ECUs.

Figure 4.6: The proposed design and implementation flow for distributed automotive CPSs using the proposed integrated toolchain.

1. Co-Flex provides a Simulink library which assists the developer to model a controller with implementation-specific details like task partitioning and mapping, task schedules, and data mapping. Thus, the developer need not model the software from scratch and, rather, can use the control blocks provided by Co-Flex and then configure the block parameters according to the specification.

2. Co-Flex offers a MATLAB toolbox that automates the manual parts in the conventional design and implementation flow, e.g., specification extraction, problem formulation, and parameter write back.

3. Co-Flex offers parameter synthesis by employing a state-of-the-art design approach that can simultaneously synthesize the controllers and the platform parameters, and in the process co-optimize the average control performance and the resource usage. The proposed approach offers design optimality and also allows the designer to make a trade-off between average control performance and resource usage.

The proposed flow for software development mainly consists of three phases. In phase (i), i.e., the *specification modeling* phase, the designer develops the application models and also adds to them the implementation-specific details, e.g., task partitioning and mapping, data mapping, and frame packing, according to the system specifications. In phase (ii), i.e., the *design and implementation* phase, the parameter synthesis problem is formulated according to the specification model from phase (i), and subsequently, the problem is solved to synthesize the controllers and the platform parameters. The specification model is then configured with the synthesized parameter values and application-specific details to complete the software model. Phase (iii) or the *code generation* phase is exactly the same as in the case of conventional flow where the C-code and binary files are generated from the software model to be flashed onto the ECUs. It may be noted that we have only replaced the first two phases of the conventional design and implementation flow with the specification modeling phase and the design and implementation phase respectively. Subsequently, in this section we will explain these two phases in further detail together with the respective tools.

4.3.1 Specification Modeling

In contrast to the conventional design and implementation flow where application software modeling starts only after the synthesis of design parameters, the proposed flow starts with modeling the system specification. For the problem setting described in Section 3.2.1, the system specification typically includes (i) plant models, (ii) type of controllers used, (iii) task partitioning and mapping, (iv) data mapping and frame packing, and (v) bus parameters.

We assume that the plant models are available and they are controlled using state-feedback controllers. The plant models are given by the continuous-time state-space matrices $\{A, B, C\}$ as defined in Eq. (2.1). For task partitioning and mapping, we assume as given: (i) the physical system layout, i.e., physical distribution of sensors, actuators, and ECUs, and (ii) the network topology, i.e., how the sensors, the actuators, the ECUs and the FlexRay bus are connected. With such assumptions, task partitioning and mapping is partially fixed, e.g., if a sensor is connected to an ECU then it makes sense to map the sensor task on the corresponding ECU. However, the rest can be synthesized by employing some well-established task partitioning and mapping algorithms [219,220]. Furthermore, from task partitioning and mapping, data dependencies can be derived, and the data that need to be transmitted as messages over the communication bus can be identified. In this work, we assume that each frame is packed with only one message, and do not consider frame packing in the design optimization. We understand that frame packing, i.e., packing of frame with more than one message, can result in a more efficient design, however, to ensure the scalability of our approach we consider frame packing as given by the specification. In addition, we assume that the FlexRay network parameters like bus cycle time T_{bus}, clock frequency, number of static slots N_s, the size of each slot Δ, among others, are fixed based on requirements, e.g., static slot size is determined by the size of the largest frame in the system. Now, given the specification, it can be modeled as a preliminary application software in Simulink Environment using *Co-Flex:Model* library and SIMTOOLS/SIMTARGET.

Co-Flex: Model – It is a Simulink library consisting of template blocks that aid the modeling of FlexRay-based control software. For the time being, the library supports modeling of feedback controllers only, however, it can be extended in future for other controllers like model predictive controllers (MPC), adaptive controllers, etc. The library mainly consists of two template blocks as follows:

- *FeedbackController*: This block accelerates modeling of a feedback control application on distributed FlexRay platform. For each control application, we need to add this block in the specification model. The parameters to this block include (i) the nomenclature of the sensor and control messages, (ii) sensor, controller and actuator task mappings and schedules, (iii) sampling period, (iv) feedback and feedforward gains (vii) control performance metric, and (viii) performance requirement. A newly added block basically represents a skeleton implementation of a distributed control application with empty sensor, controller and actuator task models. After parameterizing the block, the developer can push a button *create* (provided in the mask) to automatically configure the underlying model to represent an implementation of a distributed embedded controller. In addition, the block is reconfigurable, i.e., the developer can use the same block to model a different controller in case of change in the specification. However, if the order of the corresponding controlled plant is different then the developer must first push a button *clear* to go back to the skeleton implementation. Furthermore, a push button *verify* is provided in this block in order to check data consistency between this block and the SIMTOOLS DB block that is used for platform specification. The inputs of the block is the system state vector x and the output is the control input u.

- *Plant*: The developer can specify the plant model for a control application by parameterizing this block. It can represent a controlled plant of any order. The parameters to this block include (i) the system order, (ii) the state transition matrix, (iii) the input matrix, and (iv) the output matrix. After parameterizing the block, the developer can push a button *create* to automatically build an underlying plant model. This block is also reconfigurable and the developer can use the push button *clear* to delete the underlying model. The input of the block is the control input u and the outputs are the system state vector x and the system output y. Closed-loop system model can be obtained by connecting (i) the output x of this block to the input of the corresponding *FeedbackController* block and (ii) the output u of the corresponding *FeedbackController* block to the input of this block. The main purpose of this block is twofold: (i) the plant specification can be directly read from this block to design the corresponding controller and (ii) the underlying plant models can be used for closed-loop simulation to validate the designed controllers and platform parameters before the hardware implementation.

Now, based on *Co-Flex:Model* and SIMTOOLS/SIMTARGET, the specification modeling phase involves the following steps in order.

1. For each control application insert a *FeedbackController* and a *Plant* block, parameterize them according to the given specification, and connect them appropriately.

2. Insert a *Database File Block* provided by SIMTOOLS that allows the complete platform configuration.

3. In the *Database File Block*, configure the message signals, the ECUs and the FlexRay network according to the specification.

4. In the *Database File Block*, use *Import constraints from model* feature so that the task and message mappings are read automatically from the model.

5. Configure frames and assign messages to frames such that for each control application there are one sensor signal frame and one control signal frame.

6. Finally, the correctness of the specification model can be verified by pushing the button *verify* in the *FeedbackController* blocks.

The output of this phase is a partially specified model, where the control and schedule parameters are required to be configured and the remaining application-level details must be added in order to model the complete software implementation.

4.3.2 Design and Implementation

The main goal in this phase is to synthesize the system parameters and then further configure the specification model developed in the previous phase to represent the complete software model. This phase starts with the extraction of relevant information from the specification model which is required to formulate the parameter synthesis problem. Based on the extracted information, a co-design problem is formulated, as described in Section 3.2.2, to determine the control and platform parameters for each control application while minimizing the resource usage and maximizing the average control performance. In Chapter 3, we have proposed to solve the problem in two subsequent stages, i.e., the prospective controller design and the co-optimization. According to the proposed two-stage approach, first, for each control application, a set of prospective controllers is determined and recorded in a look-up table. Here, a prospective controller optimizes the control performance for a feasible sampling period. Particle swarm optimization (PSO) is used in combination with pole-placement for optimal controller design, as described in Section 3.3. Subsequently, we can automatically formulate a constrained multi-objective optimization problem, as given in Section 3.4, based on the extracted information from the specification model and the look-up table obtained in the previous stage. We further employ the multi-layer hybrid optimization technique to solve the problem and generate a Pareto front that comprise a number of Pareto-optimal design configurations, as described in Section 3.5. The Pareto front depicts the trade-off between the two design objectives. On the obtained Pareto front, the developer can select one of the Pareto points that is the most suitable for the overall design requirements. The parameter values corresponding to the selected Pareto point is then written back into the software model. Subsequently, the software is modeled further to incorporate application-specific details, i.e., the models for sensor and actuator tasks. The full software model can now be simulated using the plant models from the previous phase. Finally, the plant models can be removed from the software and then binary and C-code files can be generated to be flashed onto the ECUs. This aforementioned flow of the design and implementation phase can be divided into five stages, as shown in Figure 4.6, that are discussed in detail in the rest of this section.

4.3.2.1 Specification Extraction

This is the first stage of the design and implementation phase. The main essence of this stage is that it allows automated formulation of the control-platform co-design problem. In this stage, the developer can first export the platform configuration from the specification model as a Field Bus Exchange Format (FIBEX) file [221]. The *Export as FIBEX* feature is provided in the SIMTOOLS *Database File Block*. FIBEX is a standard format used by the automotive industry to exchange data. FIBEX file is typically an Extensible Markup Language (XML) document with a standard XML Schema Definition (XSD), and therefore, can be read by any standard parsing tool to extract important information. The exported FIBEX from the intermediate model contains dummy values for task and message schedules as they are not yet synthesized. The important valid information to be extracted from the exported FIBEX includes the following:

- FlexRay parameters, i.e., the bus cycle time (T_{bus}), the number of configurable communication cycles (N_{com}), the number of static slots (N_s), and the length of a static slot (Δ).
- ECU attributes, i.e., the names of the ECUs and the corresponding FlexRay controllers.
- Task attributes, i.e., the task names, the WCETs, and the task mappings.
- FlexRay frame attributes, i.e., the frame names, the mapping of data to frames, and the sizes of the constituent data.

It may be noted that the information is not complete, and the obtained FIBEX does not contain any information of the plant model, the performance metric or the performance requirement for an application, which are required to design the prospective controllers. Thus, the proposed Co-Flex MATLAB toolbox offers a tool called *Parse* that helps to extract the missing information from the specification model and also store all the relevant information in a systematic manner.

Co-Flex: Parse – *Co-Flex Toolbox* block must be inserted into the software model from which this tool can be invoked. It basically enables the specification extraction and systematic storage. Therefore, this tool parses the specification model as well as the exported FIBEX from the model. This tool uses the common MATLAB semantics (e.g., find_system and get_param) to parse the software model. It determines all the *FeedbackController* blocks inserted in the software model. Each such block represents the controller implementation for an application. Now, it can parse these blocks one by one, and from each of them, it can read the performance metric for which the corresponding controller must be optimized and the required minimum performance J_i^r. Note that these parameters are configured in these blocks during the specification modeling phase. Furthermore, it also determines the *Plant* block that is connected to a *FeedbackController* block and reads the plant model (i.e., A, B, and C matrices) and store the information in the appropriate file. On the other hand, the *Parse* tool also uses the MATLAB Document Object Model (DOM) parser to parse the FIBEX file. As discussed above, information regarding the FlexRay parameters, ECU attributes, task attributes, and FlexRay frame attributes can be extracted from the FIBEX file. This tool organizes the extracted information from the FIBEX file and the software model, as shown in Figure 4.6, into several easy-to-access csv files, i.e., *BusParam.csv*, *ECUAttr.csv*, *WCET.csv*, *TaskMap.csv*, *FrameData.csv*, and *AppX.csv* (for application *X*). These files can subsequently be used by the following stages for problem formulation as well as for writing back the synthesized parameter values into the software model.

4.3.2.2 Prospective Controller Design

This is the implementation of the first stage of the control-platform co-design approach as proposed in Chapter 3. As mentioned before, the controller and the platform design approaches have been traditionally very different in nature, and therefore, it is challenging to solve the two problems comprehensively in one step. Thus, in our proposed two-stage approach, we first design optimal controllers at all possible sampling periods for each control application. In the next stage, during the co-synthesis of control and platform parameters, the controller design basically boils down to selecting one of these prospective optimal controllers based on design objectives. In this stage, there are two main tasks that must be executed subsequently as follows:

1. From the FlexRay bus configuration, we need to determine the possible sampling periods exploiting the relation in Eq. (3.8) and the fact that FlexRay supports only a limited number of repetition rates for the messages.

2. For each application, determine the controller that maximize the control performance at each of the obtained sampling periods.

Here, Co-Flex MATLAB toolbox offers a tool called *Control* that realizes the aforementioned tasks in this stage.

Co-Flex: Control – This tool can be invoked for each application from the corresponding *FeedbackController* block. This tool first reads the file *BusParam.csv* to fetch the values of bus cycles time T_{bus} and the number of configurable communication cycle N_{com}. We know that the repetition rate of a FlexRay message can only take values in $\{2^{k-1} | k \in \mathbb{N} \wedge 1 \leq k \leq \log_2 N_{com} + 1\}$. If the sampling period of a controller is h, then the sensor and the control messages must have a repetition rate that is equal to $\frac{h}{T_{bus}}$ according to the relation in Eq. (3.8). Thus, corresponding to all possible choices for a repetition rate, we can determine the set of feasible sampling periods as $\{2^{k-1} \cdot T_{bus} | k \in \mathbb{N} \wedge 1 \leq k \leq \log_2 N_{com} + 1\}$. For example, when $N_{com} = 64$, which is a standard value for FlexRay bus configuration, the number of feasible values for the sampling period of a control application is 7.

The *Control* tool iterates through the obtained values of sampling period one by one. For a control application *X*, it fetches the plant model, the control performance metric, the required performance value from the file *AppX.csv*. For a sampling period value in the predetermined set, the *Control* tool first computes the augmented state-space model for the delayed discrete-time system based on the continuous-time plant model using Eq. (2.15) and Eq. (2.17). Note that, here, we assume that the sensing-to-actuation delay is equal to one sampling period. With these augmented state-space matrices, the sampling period and the delay, the *Control* tool uses Algorithm 2 for optimal controller design. Note that Algorithm 2 uses particle swarm optimization to effectively search the pole space while employing standard pole-placement to determine the control gains. The *Control* tool provides, as input to the algorithm, the control performance metric that needs to be optimized. Based on the input, the algorithm invokes the appropriate function to calculate the control performance for a given set of control gains (in line 10 and line 24 of Algorithm 2). The algorithm outputs the optimal normalized control performance that can be calculated based on the required value of performance as per Eq. (3.4).

The *Control* tool, therefore, formulates a look-up table for each control application C_i, as given in Figure 4.6, where for each possible value of sampling period $h^{(k)} = 2^{k-1} \cdot T_{bus}$, we can assign an optimal normalized control performance J_i^{k*} corresponding to the control gains K_i^{k*}, F_i^{k*}. In the next stage, i.e., the *co-optimization* stage, the sampling periods of all control applications will be used as variables in the optimization problem formulation. The objective of the average control performance can, therefore, be formulated as a function of the sampling periods as per Eq. (3.23). Therefore, the sampling periods here serve as the main interface for the interplay between the prospective controller design stage and the co-optimization stage. Also, given the values of the sampling periods, the control gains can be fetched from these look-up tables in the *parameter writeback* stage.

4.3.2.3 Co-Optimization

This is the second stage of the control-platform co-design approach proposed in Chapter 3. In this stage, we formulate a constrained optimization problem, as discussed in Section 3.4. The variables of the problem are the sampling periods, the task schedules including the task periods and the offsets, and the message schedules including the slot ids, the base cycles and the repetition rates. The optimization objectives are the average control performance and the resource usage as given by Eq. (3.23) and Eq. (3.24) respectively. As derived in Section 3.4.1, the problem considers the following constraints: (i) the sampling period constraints given by Eq. (3.8), (ii) the dataflow constraints given by Eq. (3.9), Eq. (3.10), Eq. (3.11), and Eq. (3.12), (iii) the sensing-to-actuation delay constraints given by Eq. (3.13), (iv) the non-overlapping tasks constraints given by Eq. (3.14), (v) the non-overlapping messages constraints given by Eq. (3.15), (vi) the FlexRay scheduling constraints given by Eq. (2.38), Eq. (3.16), and Eq. (3.17), (vii) the ECU scheduling constraints given by Eq. (3.18) and Eq. (3.19), and (viii) the performance constraints given by Eq. (3.20), Eq. (3.21), and Eq. (3.22). The constrained optimization problem formulated here for the co-design of control and platform parameters can be solved using a multi-layer hybrid optimization technique, as outlined in Section 3.5 and illustrated in Figure 3.4. Co-Flex MATLAB toolbox offers a tool named *Opti* that automates the formulation of the constrained optimization problem, and subsequently, also solves the problem.

Co-Flex: Opti: This tool can be invoked from the *Co-Flex Toolbox* block in the software model. It synthesizes valid Pareto-optimal design configurations. Towards this, it implements the proposed multi-layer hybrid optimization technique. The implementation comprises three functions as follows:

- The first function implements the outer layer of the optimization as given in Algorithm 3. This function requires as input the look-up tables obtained from the prospective controller design stage. For each application, it, therefore, determines the maximum and the minimum sampling periods for which optimal controllers can be respectively designed such that the normalized control performance is less than or equal to 100%. Accordingly, the function calculates the minimum U^- and the maximum U^+ possible resource usage using

Eq. (3.25) and Eq. (3.26). It further calculates the minimum difference U^{Δ} between two feasible values of resource usage as per Eq. (3.27). Now, it can iterate through feasible values of resource usage from U^{-} to U^{+} with a step of U^{Δ}. For each value of resource usage U^{*}, this function calls the inner layers to check if there exists a Pareto-optimal design configuration corresponding to U^{*}.

- The second function implements the middle layer of the optimization. It formulates an MILP problem that minimizes the average control performance while satisfying an equality constraint on resource usage, and in the process, determines the sampling periods of the applications. Here, the function reads J_i^{k*}s from the look-up tables obtained from the prospective controller design stage and the bus parameters, T_{bus} and N_{com}, in *BusParam.csv* obtained from the specification extraction stage. Considering that there are $1 + \log_2 N_{com}$ choices of sampling periods, we need to consider $1 + \log_2 N_{com}$ boolean variables for each application. Correspondingly, the equality constraint on resource usage can be derived from Eq. (3.24) as follows:

$$U^{*} = \sum_{i=1}^{|\mathcal{C}|} \sum_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} \cdot 2^{\log_2 N_{com}+2-k}. \tag{4.1}$$

That is, when the sampling period $h_i = 2^{k-1} \cdot T_{bus}$, the number of slots used by the messages of the application C_i in a series of N_{com} cycles is equal to $2^{\log_2 N_{com}+2-k}$. Furthermore, we need to consider a constraint on the boolean variables such that only one of them can be 1 for an application. This is written as follows:

$$\forall_{C_i \in \mathcal{C}} \left(\sum_{k=1}^{1+\log_2 N_{com}} \psi_{i,k} = 1 \right). \tag{4.2}$$

The MILP also needs to consider the performance constraint in Eq. (3.22). In addition, the MILP need to consider a constraint given by:

$$J^{-} < J^{*}_{av} < J^{+}. \tag{4.3}$$

Here, J^{+} ensures Pareto optimality of the solution while J^{-} allows to search for Pareto candidates even when the best candidate is not feasible. This is explained in more detail in Section 3.5.2. The objective is to minimize the linear expression in Eq. (3.23). The solution to this problem will provide the values of the binary variables that can be used to calculate the values of the sampling periods using the value of T_{bus} as per Eq. (3.20). Note that according to Algorithm 4, all the solutions of the MILP problem need to be determined. Here, we employ CPLEX [171] to solve the MILP problem, where the 'Solution Pool' feature enables to determine all possible optimal solutions. Now, for each optimal set of sampling periods obtained as a solution, this function invokes the inner layer until a Pareto point is found.

- The third function implements the inner layer where an ILP is formulated to determine the unknown parameters of task and message schedules while considering all platform constraints. According to the sampling period constraint in Eq. (3.8), we can determine the task periods and the message repetition rates using the obtained values of sampling periods from the middle layer. Here, the value of T_{bus} is read from *BusParam.csv*. Now, the remaining unknowns are the offsets of the tasks, the slot ids and the base cycles of the messages. The constraints of the ILP are the dataflow constraints, the sensing-to-actuation delay constraints, the non-overlapping tasks constraints, the non-overlapping messages constraints, the FlexRay scheduling constraints, and the ECU scheduling constraints. Note that all constraints are linear. SIMTOOLS allows configuration of timing parameters with the granularity of one microsecond, and therefore, the variables of the ILP are all integers. To formulate the constraints, the function reads the values of the constant parameters from the files *TaskWCET.csv* and *BusParam.csv*. Furthermore, for non-overlapping tasks constraints, the function reads the task mappings from *TaskMap.csv*. Note that the ILP does not have an objective function. Here, we use Gurobi [172] to solve the ILP. If there exists a feasible solution to the ILP, then this solution is combined with the given values of the sampling periods, the task periods, and the message repetition rates to form a Pareto point.

4.3.2.4 Parameter Writeback

The Pareto front thus obtained from the co-optimization stage consists of a number of Pareto points where each represents a feasible Pareto optimal solution of the co-optimization problem. The developer can choose one of these design configurations to be implemented based on design requirements. The optimal solution corresponding to the Pareto point must be interpreted correctly to represent a valid design configuration. Thereafter, the specification model must be parameterized accordingly. To facilitate convenient write back of the synthesized parameter values, Co-flex MATLAB toolbox offers a tool named *Writeback*.

Co-Flex: Writeback – This tool can be invoked from the *Co-Flex Toolbox* block in the software model. It automates (i) completely the result interpretation and (ii) partially the result write back. The solution to the co-optimization problem is stored as a matrix where each row represents a Pareto point. Therefore, if the developer selects the n-th Pareto point then the *Writeback* tool will extract the n-th row from the solution matrix. The elements in the extracted row can be interpreted to determine the sampling periods, the task and the message schedules for the control application. Note that the WCET of a task, which is also a schedule parameter, is already configured in the specification modeling stage. The result interpretation is performed by exploiting the knowledge of the problem formulation from the previous stage, i.e., if a design parameter is represented by the decision variable i then the synthesized value of the parameter is the value corresponding to the i-th element of the extracted row. On the other hand, as shown in Figure 4.6, the synthesized values of control gains can be obtained by this tool from the look-up tables generated in the *Prospective Control Design* stage according to the synthesized values of the sampling periods.

Following result interpretation, the developer needs to configure the software model with the synthesis result, which is realized as follows:

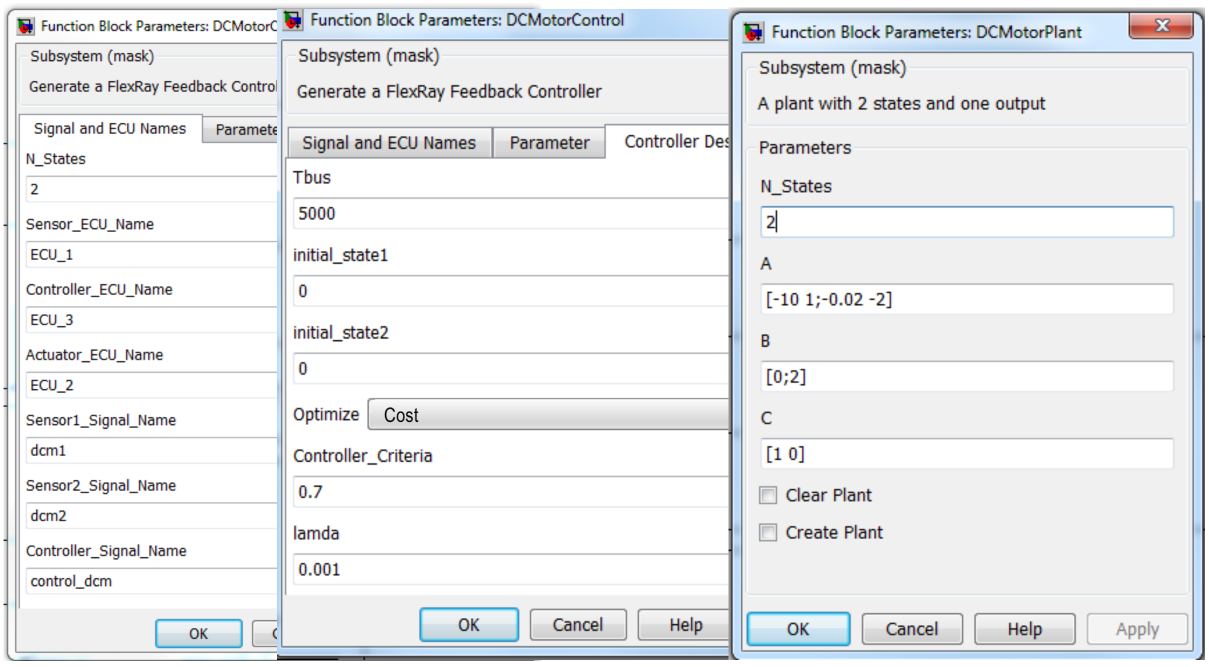
- The *Writeback* tool can directly write the control gains as parameters to the *Feedback-Controller*. The underlying controller model are configured with the gain values automatically.
- The application task schedules are also written by the *Writeback* tool as parameters to the corresponding *FeedbackController* block. The underlying controller implementation comprises *Dispatch Event* block for each task (i.e., sensor, controller and actuator tasks). These blocks takes the value of the task offset and the task period automatically from the parameters of the *FeedbackController* block.
- For the FlexRay frame schedules, the *Writeback* tool generates a csv file in the format which can be directly imported in the SIMTOOLS *Database File Block*. The format also includes other attributes of the frames like the transmitting and receiving FlexRay controller and frame size, that can be obtained by referring to *FrameData.csv* generated in the *Specification Extraction* stage.
- For the communication tasks, schedules are generated as a csv file in a format in which it is to be entered in the SIMTOOLS *Database File Block*. However, SIMTOOLS does not allow importing communication task schedules, and therefore, they are manually entered by the developer according to the generated file. In our opinion, this process can be made completely automated by collaborating with tool suppliers.

Following the parameter write back, the developed software model can be tested via closed-loop simulation using the existing plant models. In this regard, SIMTOOLS offers a *Simulation Configuration Block* that allows to verify timing accuracy in addition to functional correctness.

4.3.2.5 Application Software Modeling

The software implementation developed so far contains *Plant* blocks that are used only for the design and simulation. They are not part of the final software that will run on the ECUs. In the code generation phase later, we use SIMTOOLS *Split and Build* function to partition the whole software model into several components, where each component represents the software model for an ECU, from which codes can be generated for the ECU. Now, since the *Split and Build* function does not look inside a subsystem block, the controller implementation that lie inside the *FeedbackController* block must be brought to the first level for correct generation of codes and binaries in the next phase. Co-Flex MATLAB toolbox offers a tool named *Dissemble* that automates these two required steps.

Co-Flex: Dissemble – This tool can be invoked from the *Co-Flex Toolbox* block. It takes the development closer to the implementation by (i) deleting the plant models and (ii) expanding subsystem blocks that represent the controller implementations. However, it might be required to reuse the model developed so far partially or completely for a different car with different requirements. Therefore, it makes sense not to modify the software model developed so far and rather create a new model by copying only the parts that will be implemented. The *Dissemble* tool automatically copies the blocks used in controller implementations and the platform configuration block (i.e., the SIMTOOLS' *Database File Block*) into a new Simulink model



Function Block Parameters: DCMotorC
Subsystem (mask)
Generate a FlexRay Feedback Control
Signal and ECU Names | Paramete
N_States
2
Sensor_ECU_Name
ECU_1
Controller_ECU_Name
ECU_3
Actuator_ECU_Name
ECU_2
Sensor1_Signal_Name
dcm1
Sensor2_Signal_Name
dcm2
Controller_Signal_Name
control_dcm
OK

Function Block Parameters: DCMotorControl
Subsystem (mask)
Generate a FlexRay Feedback Controller
Signal and ECU Names | Parameter | Controller Des
Tbus
5000
initial_state1
0
initial_state2
0
Optimize: Cost
Controller_Criteria
0.7
lamda
0.001
OK | Cancel | Help

Function Block Parameters: DCMotorPlant
Subsystem (mask)
A plant with 2 states and one output
Parameters
N_States
2
A
[-10 1;-0.02 -2]
B
[0;2]
C
[1 0]
☐ Clear Plant
☐ Create Plant
OK | Cancel | Help | Apply

Figure 4.7: Specification modeling using Co-Flex. (From left to right) (i) Task mapping and signal definition; (ii) Control requirements; (iii) Plant model.

and also adds blocks for clock synchronization. This can be achieved by using the MATLAB function *add_block*.

Furthermore, the implementation so far may not contain the detailed model of the sensor and the actuator tasks. These models depend on the type of sensors and actuators used. For example, in case of adaptive cruise control, the sensor can be a camera and the sensor task requires image processing to detect the braking of the vehicle ahead. Therefore, in this stage, the developer needs to manually incorporate these task models into the software. In future, some standard task models can also be added to *Co-Flex: Model* Simulink library to further reduce manual intervention. This final software with complete application models is then used in the next phase for code generation and hardware implementation.

4.4 A Case Study

In this section, we consider the same case study as in Section 3.6.1 that comprises five control applications, namely, a DC motor speed control (DCM), a car suspension system (CSS), an electronic wedge brake (EWB), and two variants of cruise control (CC1 and CC2 respectively). These applications run on three ECUs, namely, ECU_1, ECU_2, and ECU_3 respectively. Here, we will outline how we can develop the control software for this system using the proposed integrated toolchain and by following the suggested development process step-by-step.

Specification modeling: We first create a specification model as described in Section 4.3.1. Here, we first start a new Simulink model and for each application, we drag and drop a *Feed-*

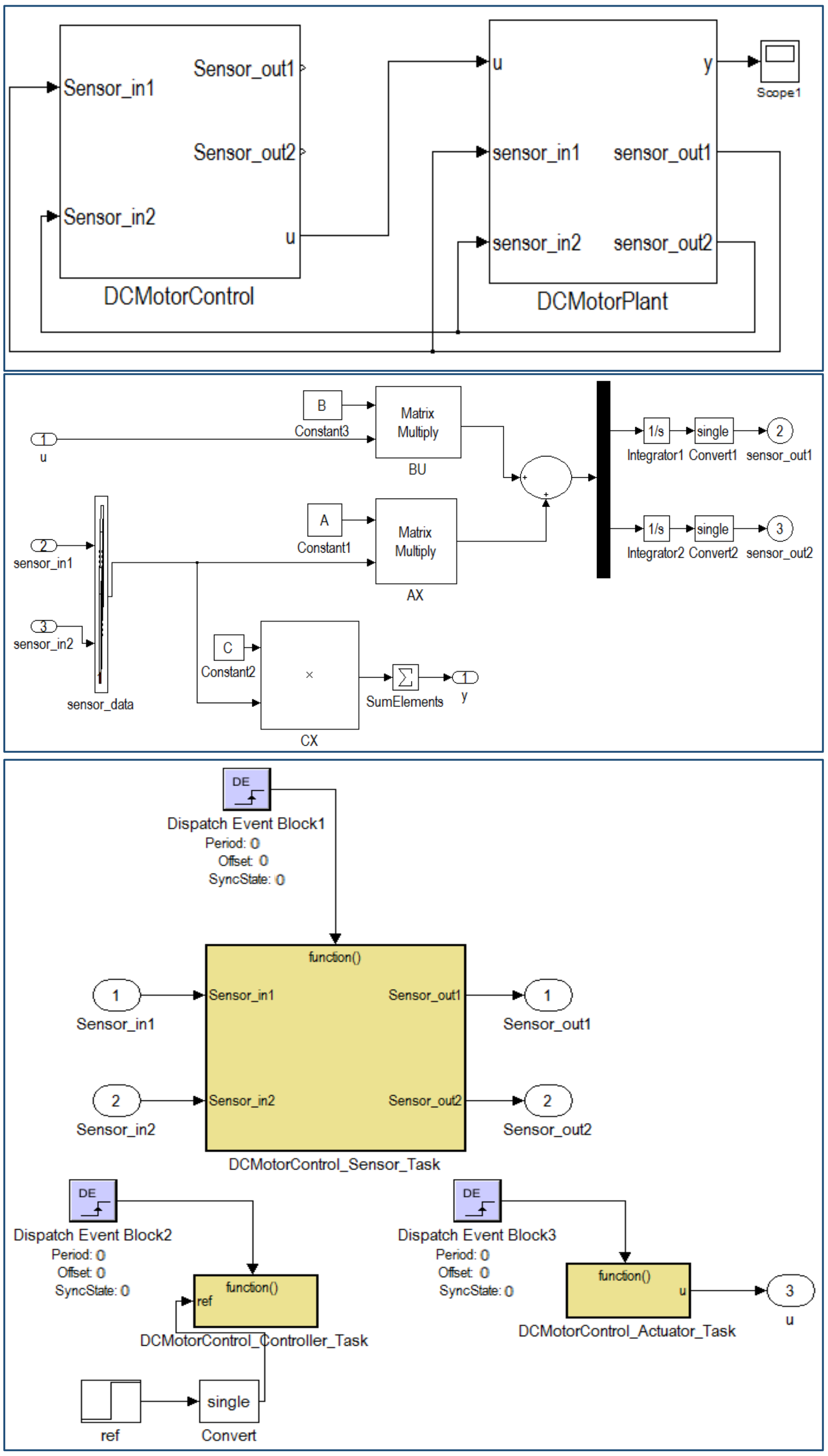
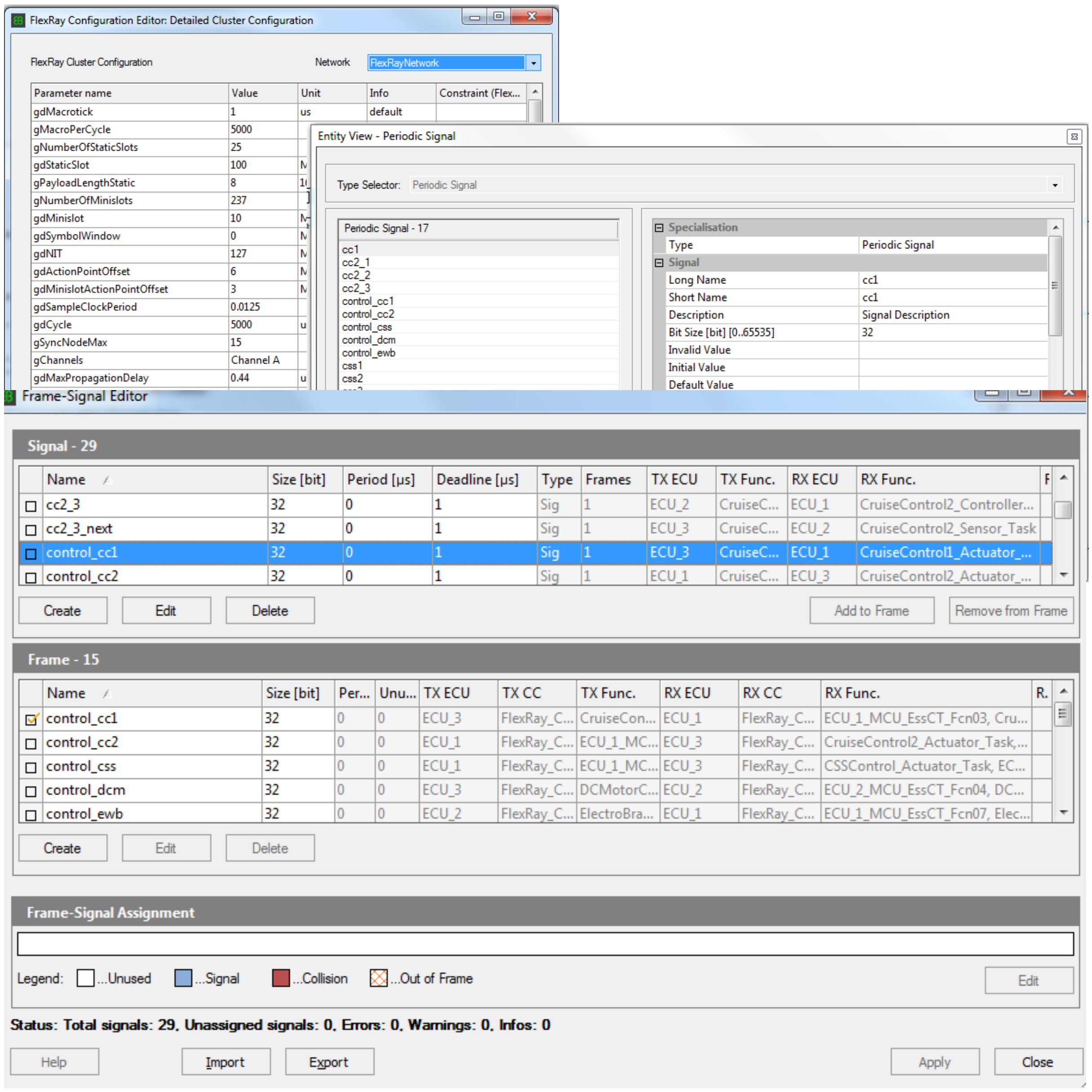


Figure 4.8: Partially specified model. (i) Closed-loop system model (top); (ii) Plant model (middle); (iii) Distributed controller implementation (bottom).



FlexRay Configuration Editor: Detailed Cluster Configuration

FlexRay Cluster Configuration   Network: FlexRayNetwork

Parameter name	Value	Unit	Info	Constraint (Flex...
gdMacrotick	1	us	default	
gMacroPerCycle	5000			
gNumberOfStaticSlots	25			
gdStaticSlot	100	M		
gPayloadLengthStatic	8	1(		
gNumberOfMinislots	237	]		
gdMinislot	10	M		
gdSymbolWindow	0	M		
gdNIT	127	M		
gdActionPointOffset	6	M		
gdMinislotActionPointOffset	3	M		
gdSampleClockPeriod	0.0125			
gdCycle	5000	u		
gSyncNodeMax	15			
gChannels	Channel A			
gdMaxPropagationDelay	0.44	u		

Entity View - Periodic Signal

Type Selector: Periodic Signal

Periodic Signal - 17

cc1
cc2_1
cc2_2
cc2_3
control_cc1
control_cc2
control_css
control_dcm
control_ewb
css1
css2

Specialisation	
Type	Periodic Signal
Signal	
Long Name	cc1
Short Name	cc1
Description	Signal Description
Bit Size [bit] [0..65535]	32
Invalid Value	
Initial Value	
Default Value	

Frame-Signal Editor

Signal - 29

	Name	Size [bit]	Period [µs]	Deadline [µs]	Type	Frames	TX ECU	TX Func.	RX ECU	RX Func.	F
☐	cc2_3	32	0	1	Sig	1	ECU_2	CruiseC...	ECU_1	CruiseControl2_Controller...	
☐	cc2_3_next	32	0	1	Sig	1	ECU_3	CruiseC...	ECU_2	CruiseControl2_Sensor_Task	
☐	control_cc1	32	0	1	Sig	1	ECU_3	CruiseC...	ECU_1	CruiseControl1_Actuator_...	
☐	control_cc2	32	0	1	Sig	1	ECU_1	CruiseC...	ECU_3	CruiseControl2_Actuator_...	

Create   Edit   Delete   Add to Frame   Remove from Frame

Frame - 15

	Name	Size [bit]	Per...	Unu...	TX ECU	TX CC	TX Func.	RX ECU	RX CC	RX Func.	R.
☑	control_cc1	32	0	0	ECU_3	FlexRay_C...	CruiseCon...	ECU_1	FlexRay_C...	ECU_1_MCU_EssCT_Fcn03, Cru...	
☐	control_cc2	32	0	0	ECU_1	FlexRay_C...	ECU_1_MC...	ECU_3	FlexRay_C...	CruiseControl2_Actuator_Task,...	
☐	control_css	32	0	0	ECU_1	FlexRay_C...	ECU_1_MC...	ECU_3	FlexRay_C...	CSSControl_Actuator_Task, EC...	
☐	control_dcm	32	0	0	ECU_3	FlexRay_C...	DCMotorC...	ECU_2	FlexRay_C...	ECU_2_MCU_EssCT_Fcn04, DC...	
☐	control_ewb	32	0	0	ECU_2	FlexRay_C...	ElectroBra...	ECU_1	FlexRay_C...	ECU_1_MCU_EssCT_Fcn07, Elec...	

Create   Edit   Delete

Frame-Signal Assignment

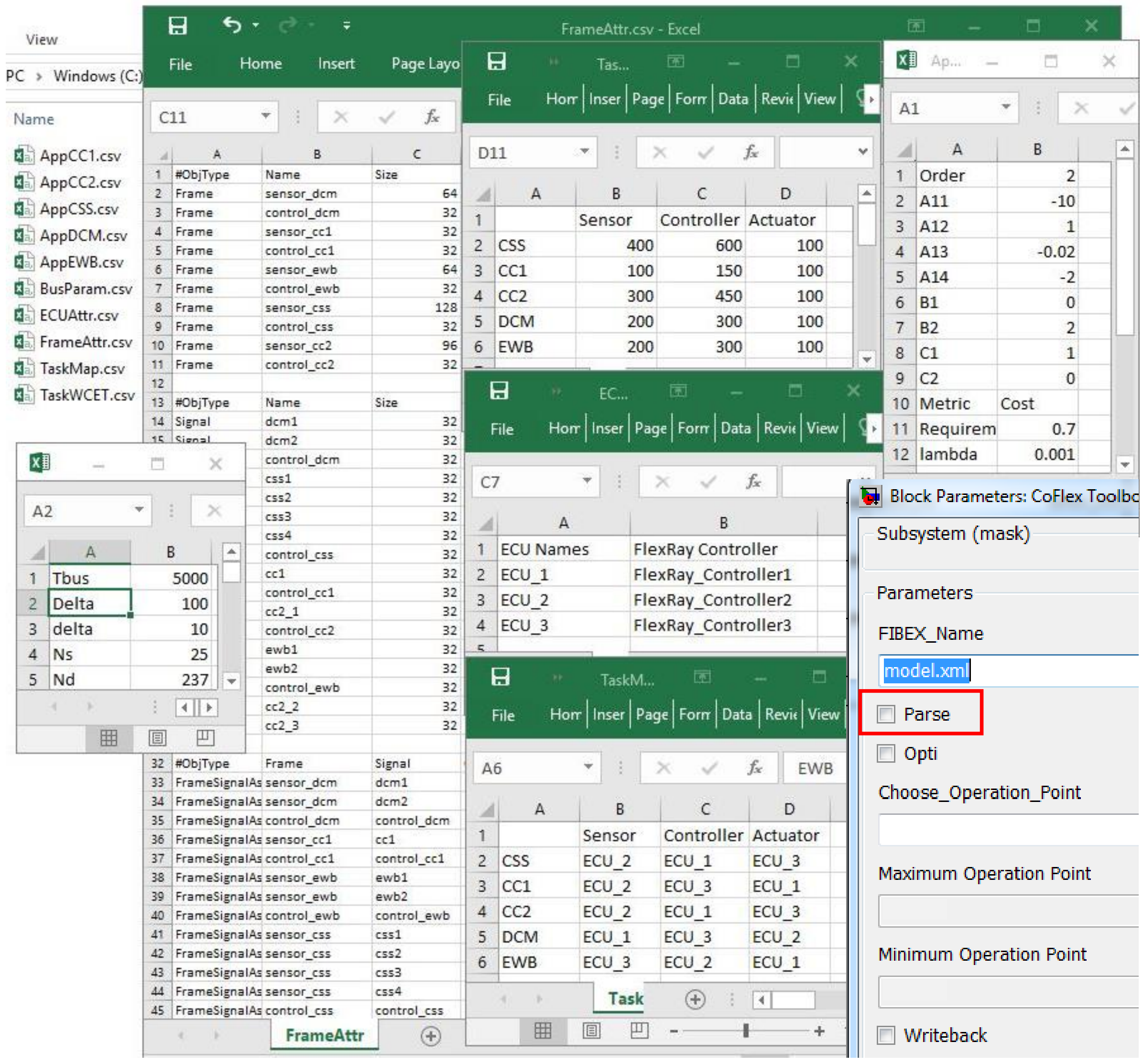
Legend: ☐ ...Unused 🟦 ...Signal 🟥 ...Collision ☒ ...Out of Frame   Edit

Status: Total signals: 29, Unassigned signals: 0, Errors: 0, Warnings: 0, Infos: 0

Help   Import   Export   Apply   Close

Figure 4.9: Specification modeling using SIMTOOLS. (i) FlexRay configuration (top left); (ii) Signal description (top right); (iii) Data mapping and frame packing (bottom).

backController block and a *Plant* block from the *Co-Flex: Model* library. These blocks corresponding to the DCM application are shown in the top snapshot in Figure 4.8. Now, we configure these blocks as per the specification. As shown in the left snapshot in Figure 4.7, the task mappings and the signal names are specified as parameters to the *FeedbackController* block. Similarly, the control requirements captured by the performance metric, the required value of the metric, and the value of λ can also be provided in the *FeedbackController* block as shown in the middle snapshot in Figure 4.7. Furthermore, the right snapshot in Figure 4.7 shows how we can specify the plant model in the *Plant* block including the order and the continuous-time system matrices. Based on this parametrization, the underlying model of the plant and the



FrameAttr.csv (C11)

	A	B	C
1	#ObjType	Name	Size
2	Frame	sensor_dcm	64
3	Frame	control_dcm	32
4	Frame	sensor_cc1	32
5	Frame	control_cc1	32
6	Frame	sensor_ewb	64
7	Frame	control_ewb	32
8	Frame	sensor_css	128
9	Frame	control_css	32
10	Frame	sensor_cc2	96
11	Frame	control_cc2	32
12			
13	#ObjType	Name	Size
14	Signal	dcm1	32
15	Signal	dcm2	32
		control_dcm	32
		css1	32
		css2	32
		css3	32
		css4	32
		control_css	32
		cc1	32
		control_cc1	32
		cc2_1	32
		control_cc2	32
		ewb1	32
		ewb2	32
		control_ewb	32
		cc2_2	32
		cc2_3	32
32	#ObjType	Frame	Signal
33	FrameSignalAs	sensor_dcm	dcm1
34	FrameSignalAs	sensor_dcm	dcm2
35	FrameSignalAs	control_dcm	control_dcm
36	FrameSignalAs	sensor_cc1	cc1
37	FrameSignalAs	control_cc1	control_cc1
38	FrameSignalAs	sensor_ewb	ewb1
39	FrameSignalAs	sensor_ewb	ewb2
40	FrameSignalAs	control_ewb	control_ewb
41	FrameSignalAs	sensor_css	css1
42	FrameSignalAs	sensor_css	css2
43	FrameSignalAs	sensor_css	css3
44	FrameSignalAs	sensor_css	css4
45	FrameSignalAs	control_css	control_css

Tas... (D11)

	A	B	C	D
1		Sensor	Controller	Actuator
2	CSS	400	600	100
3	CC1	100	150	100
4	CC2	300	450	100
5	DCM	200	300	100
6	EWB	200	300	100

Ap... (A1)

	A	B
1	Order	2
2	A11	-10
3	A12	1
4	A13	-0.02
5	A14	-2
6	B1	0
7	B2	2
8	C1	1
9	C2	0
10	Metric	Cost
11	Requirem	0.7
12	lambda	0.001

(A2)

	A	B
1	Tbus	5000
2	Delta	100
3	delta	10
4	Ns	25
5	Nd	237

EC... (C7)

	A	B
1	ECU Names	FlexRay Controller
2	ECU_1	FlexRay_Controller1
3	ECU_2	FlexRay_Controller2
4	ECU_3	FlexRay_Controller3

TaskM... (A6: EWB)

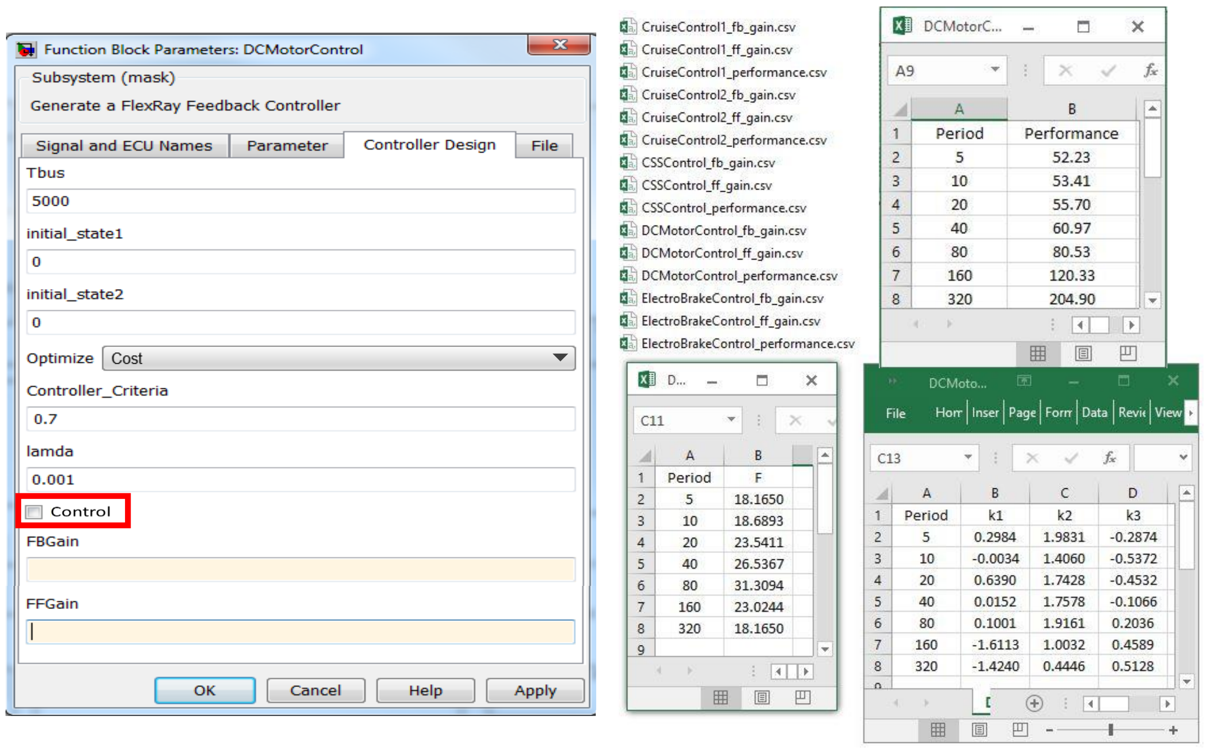
	A	B	C	D
1		Sensor	Controller	Actuator
2	CSS	ECU_2	ECU_1	ECU_3
3	CC1	ECU_2	ECU_3	ECU_1
4	CC2	ECU_2	ECU_1	ECU_3
5	DCM	ECU_1	ECU_3	ECU_2
6	EWB	ECU_3	ECU_2	ECU_1

FIBEX_Name: model.xml
☐ Parse
☐ Opti
Choose_Operation_Point:
Maximum Operation Point:
Minimum Operation Point:
☐ Writeback

Figure 4.10: Specification extraction: *Co-Flex: Parse* tool is invoked by checking the checkbox as marked by the red box. This tool reads the partially specified software model and the FIBEX file imported from the *Database File Block* to organize relevant information into several CSV files (as shown). These files can be read to formulate the control-platform co-design problem.

controller are automatically generated. The middle snapshot in Figure 4.8 shows the simulation model for the second order plant corresponding to the DCM application. The bottom snapshot in Figure 4.8 illustrates the automatically generated distributed implementation of the DCM application comprising the sensor, the controller, and the actuator tasks. In this snapshot, we can also see that the tasks are dispatched in a time-triggered manner using the respective *Dispatch Event Blocks*.

Now, we drag and drop a *Database File Block* into the Simulink model from the SIMTOOLS library. In this block, we provide the platform details as shown in Figure 4.9. Here, we can first specify the FlexRay parameters given in Table 3.3 as shown in the top left snapshot in the figure.


Function Block Parameters: DCMotorControl

Subsystem (mask)

Generate a FlexRay Feedback Controller

Signal and ECU Names | Parameter | Controller Design | File

Tbus: 5000

initial_state1: 0

initial_state2: 0

Optimize: Cost

Controller_Criteria: 0.7

lamda: 0.001

☐ Control

FBGain:

FFGain:

OK | Cancel | Help | Apply

CruiseControl1_fb_gain.csv
CruiseControl1_ff_gain.csv
CruiseControl1_performance.csv
CruiseControl2_fb_gain.csv
CruiseControl2_ff_gain.csv
CruiseControl2_performance.csv
CSSControl_fb_gain.csv
CSSControl_ff_gain.csv
CSSControl_performance.csv
DCMotorControl_fb_gain.csv
DCMotorControl_ff_gain.csv
DCMotorControl_performance.csv
ElectroBrakeControl_fb_gain.csv
ElectroBrakeControl_ff_gain.csv
ElectroBrakeControl_performance.csv

DCMotorC... (A9)

	A	B
1	Period	Performance
2	5	52.23
3	10	53.41
4	20	55.70
5	40	60.97
6	80	80.53
7	160	120.33
8	320	204.90

D... (C11)

	A	B
1	Period	F
2	5	18.1650
3	10	18.6893
4	20	23.5411
5	40	26.5367
6	80	31.3094
7	160	23.0244
8	320	18.1650
9		

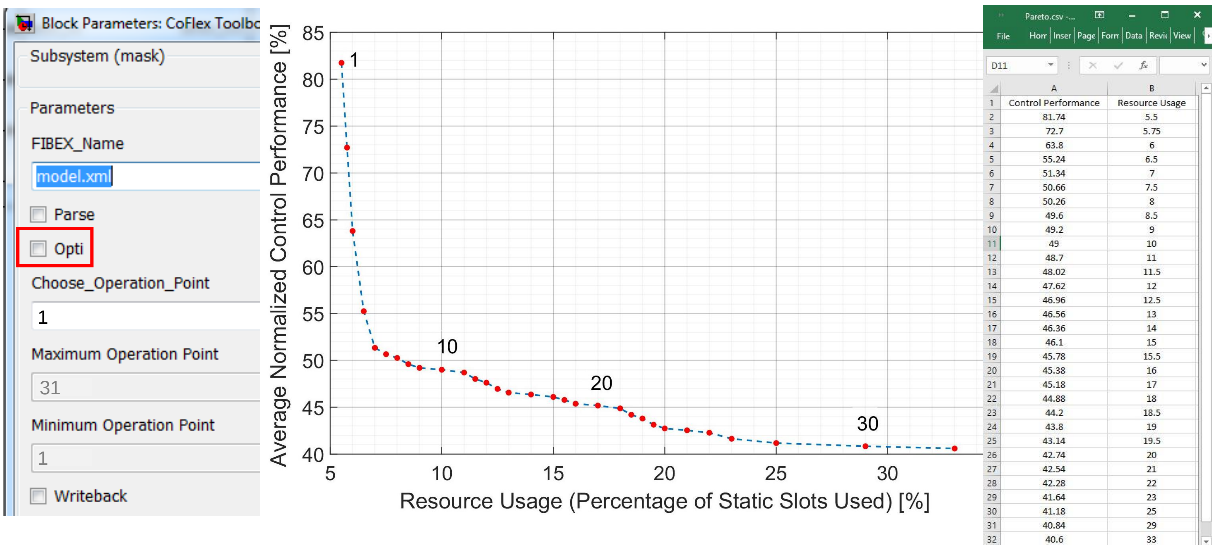
DCMoto... (C13)

	A	B	C	D
1	Period	k1	k2	k3
2	5	0.2984	1.9831	-0.2874
3	10	-0.0034	1.4060	-0.5372
4	20	0.6390	1.7428	-0.4532
5	40	0.0152	1.7578	-0.1066
6	80	0.1001	1.9161	0.2036
7	160	-1.6113	1.0032	0.4589
8	320	-1.4240	0.4446	0.5128

Figure 4.11: Prospective controller design: *Co-Flex: Control* tool (marked by a red box) can be invoked for the DCM application from the corresponding *FeedbackController* block. The feedback and the feedforward gains and the control performances of the prospective optimal controllers are tabulated in three separate CSV files (as shown). For all applications, such files are generated at the end of this stage.

We further define signals corresponding to the sensor and control data that will be sent over the FlexRay bus, as shown in the top right snapshot in the figure. Note that for a third-order plant, e.g., CC2, we need to define three signals for the three states of the plant that will be read periodically by the sensors. Next, we map signals to frames as shown in the bottom snapshot in the figure. Note that only one sensor message carries all sensor signals from the sensor task to the controller task and the signal mapping must be done accordingly. Furthermore, the details of the tasks can be automatically imported from the Simulink model into the *Database File Block* using an option offered by SIMTOOLS.

Specification extraction: We have a partially configured software model after the specification modeling phase where certain parameters, e.g., the task and message schedules and the control gains, are yet to be configured. In the design and implementation phase, we synthesize these parameters. Towards this, in the specification extraction stage (as described in Section 4.3.2.1), we first export a FIBEX file containing the data entered in the *Database File Block*. Then, we drag and drop the *Co-Flex: Toolbox* block into the Simulink model and invoke the *Co-Flex: Parse* tool from the block by checking the corresponding checkbox, as shown in Figure 4.10 (marked by a red box). This tool parses the Simulink model as well as the exported FIBEX file to extract relevant information for the formulation of the control-platform co-design problem



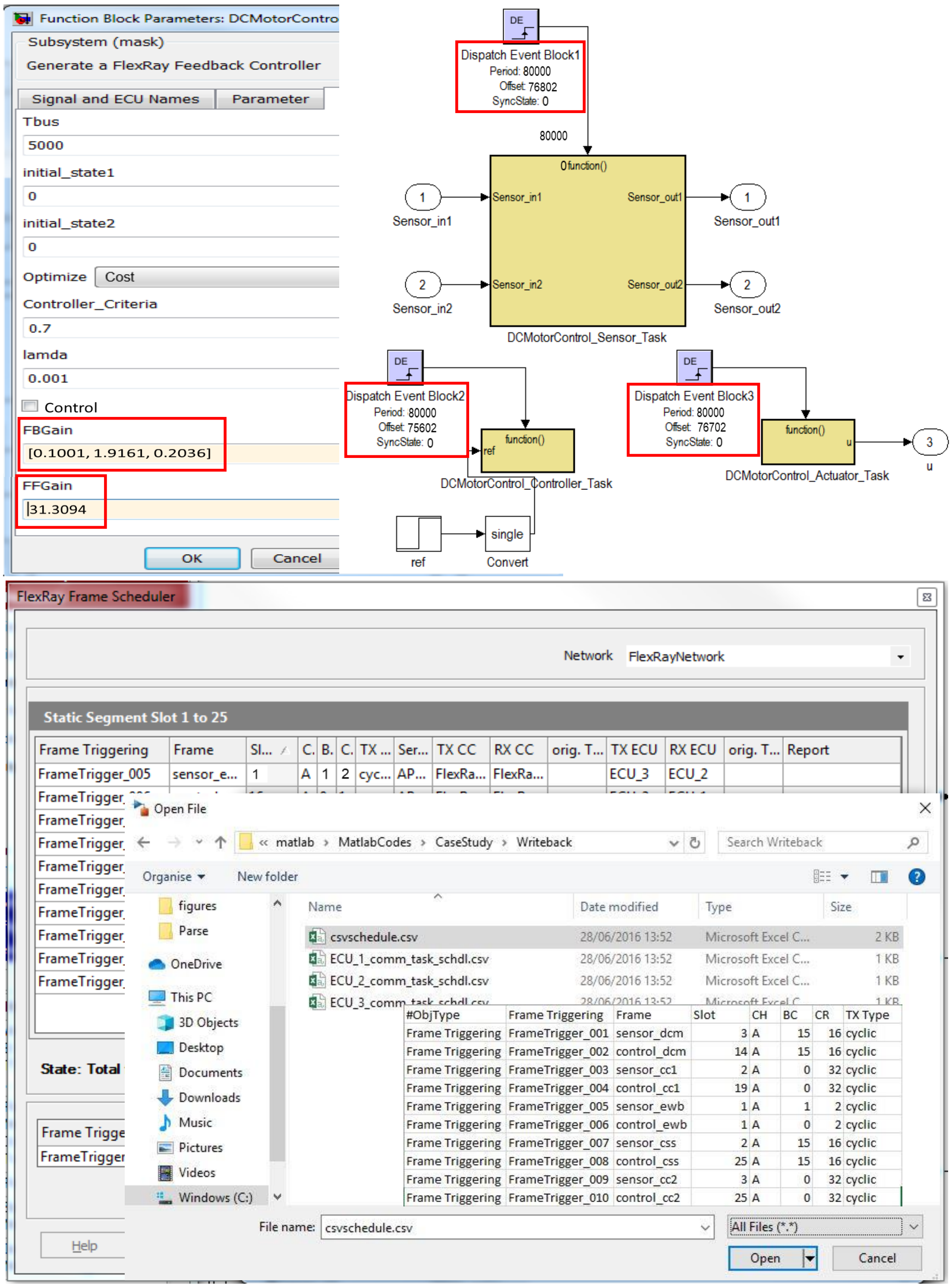
	A	B
1	Control Performance	Resource Usage
2	81.74	5.5
3	72.7	5.75
4	63.8	6
5	55.24	6.5
6	51.34	7
7	50.66	7.5
8	50.26	8
9	49.6	8.5
10	49.2	9
11	49	10
12	48.7	11
13	48.02	11.5
14	47.62	12
15	46.96	12.5
16	46.56	13
17	46.36	14
18	46.1	15
19	45.78	15.5
20	45.38	16
21	45.18	17
22	44.88	18
23	44.2	18.5
24	43.8	19
25	43.14	19.5
26	42.74	20
27	42.54	21
28	42.28	22
29	41.64	23
30	41.18	25
31	40.84	29
32	40.6	33

Figure 4.12: Co-optimization: *Co-Flex: Opti* tool (marked by the red box) can be invoked from the *Co-Flex Toolbox* block by checking the checkbox (left). It generates the Pareto front by running the co-optimization algorithm and stores it as an image (center). It further generates a CSV file (right) that stores the resource usage and the average control performance corresponding to each Pareto point.

and organize them into several machine readable files as shown in Figure 4.10. These files contain details of the FlexRay configuration, signal and frame attributes and the mapping of signals to frames, task WCETs, names of the ECUs and the corresponding FlexRay controllers, task mapping, and plant models and control requirements.

Prospective controller design: In this stage, prospective optimal controllers are designed for each application at each possible sampling period. *Co-Flex: Control* tool can be invoked for each application from the corresponding *FeedbackController* block. In the left snapshot in Figure 4.11, it is shown that for the DCM application, we can invoke the *Co-Flex: Control* tool by checking a checkbox (maked by a red box) under the “Controller Design” tab in the DCM *FeedbackController* block. This tool reads the details of the FlexRay configuration to determine the choices of sampling period. It further reads the plant models and the control requirements for the DCM application from the file generated by the *Co-Flex: Parse* tool. Now, it runs Algorithm 2 for each possible sampling period to determine the optimal feedback and feedforward control gains and the optimal control performance. At the end, this tool generate files to store the gain and the performance values corresponding to all possible sampling periods as shown in Figure 4.11.

Co-optimization: Now, we have all information required to formulate the control-platform co-design problem as proposed in Chapter 3. We can invoke the *Co-Flex: Opti* tool from the *Co-Flex Toolbox* block as shown in the left snapshot in Figure 4.12. This tool executes the nested three layer optimization approach described in Section 3.5 and generates a Pareto front as shown in the middle snapshot in Figure 4.12. This Pareto front is stored as an image file that the user can view to evaluate the trade-off opportunity for a particular problem. In



Function Block Parameters: DCMotorContro

Subsystem (mask)

Generate a FlexRay Feedback Controller

Signal and ECU Names | Parameter

Tbus	5000
initial_state1	0
initial_state2	0
Optimize	Cost
Controller_Criteria	0.7
lamda	0.001
Control	☐
FBGain	[0.1001, 1.9161, 0.2036]
FFGain	31.3094

FlexRay Frame Scheduler

Network: FlexRayNetwork

Static Segment Slot 1 to 25

Frame Triggering	Frame	Sl...	C.	B.	C.	TX ...	Ser...	TX CC	RX CC	orig. T...	TX ECU	RX ECU	orig. T...	Report
FrameTrigger_005	sensor_e...	1	A	1	2	cyc...	AP...	FlexRa...	FlexRa...		ECU_3	ECU_2		

State: Total

Open File

« matlab › MatlabCodes › CaseStudy › Writeback

Name	Date modified	Type	Size
csvschedule.csv	28/06/2016 13:52	Microsoft Excel C...	2 KB
ECU_1_comm_task_schdl.csv	28/06/2016 13:52	Microsoft Excel C...	1 KB
ECU_2_comm_task_schdl.csv	28/06/2016 13:52	Microsoft Excel C...	1 KB
ECU_3_comm_task_schdl.csv	28/06/2016 13:52	Microsoft Excel C...	1 KB

#ObjType	Frame Triggering	Frame	Slot	CH	BC	CR	TX Type
Frame Triggering	FrameTrigger_001	sensor_dcm	3	A	15	16	cyclic
Frame Triggering	FrameTrigger_002	control_dcm	14	A	15	16	cyclic
Frame Triggering	FrameTrigger_003	sensor_cc1	2	A	0	32	cyclic
Frame Triggering	FrameTrigger_004	control_cc1	19	A	0	32	cyclic
Frame Triggering	FrameTrigger_005	sensor_ewb	1	A	1	2	cyclic
Frame Triggering	FrameTrigger_006	control_ewb	1	A	0	2	cyclic
Frame Triggering	FrameTrigger_007	sensor_css	2	A	15	16	cyclic
Frame Triggering	FrameTrigger_008	control_css	25	A	15	16	cyclic
Frame Triggering	FrameTrigger_009	sensor_cc2	3	A	0	32	cyclic
Frame Triggering	FrameTrigger_010	control_cc2	25	A	0	32	cyclic

File name: csvschedule.csv

All Files (*.*)

Figure 4.13: Parameter write back: The control gains (top left) and the task schedules (top right) are directly written into the model. A CSV file is generated containing the frame schedules that can be directly imported into the *Database File Block*.

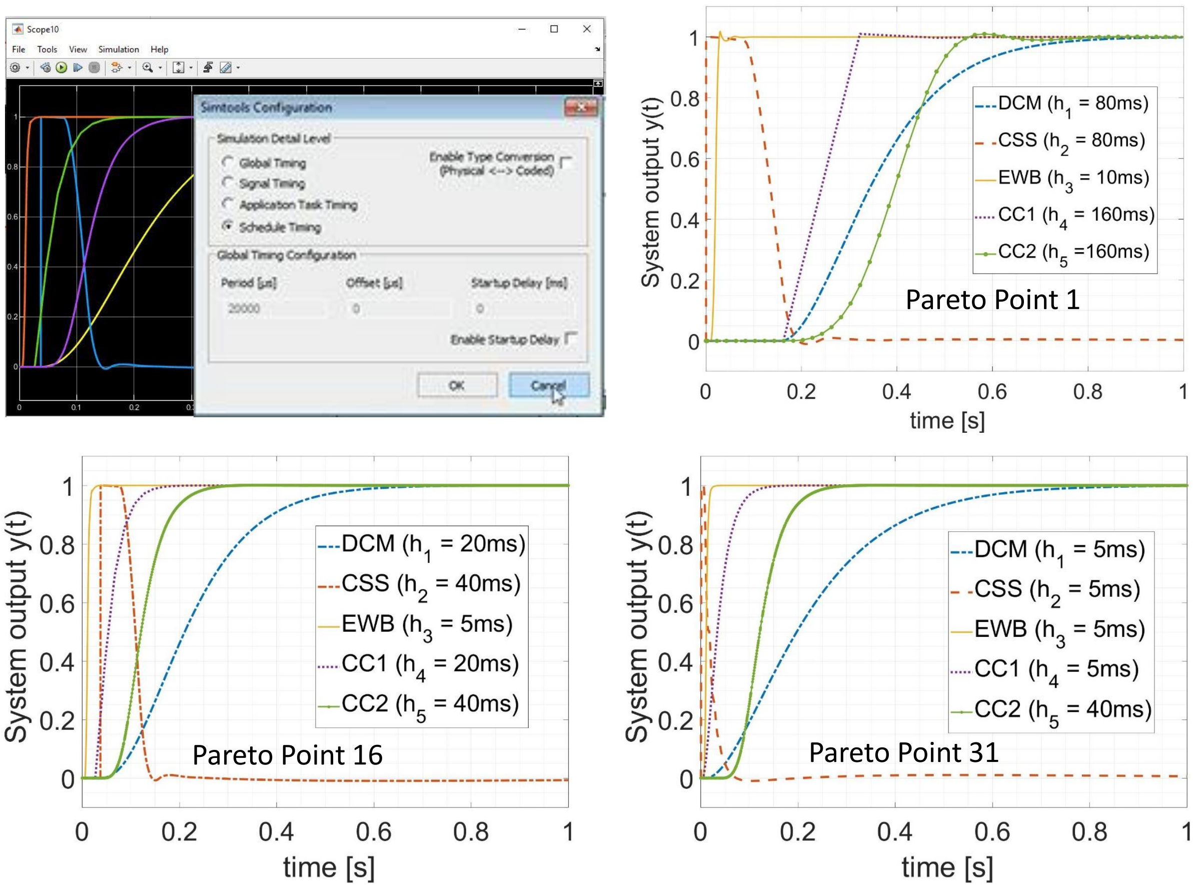
addition, this tool also generates a CSV file that stores the values of the resource usage and the average control performance for each Pareto point as shown in the right snapshot in Figure 4.12. The developer can refer to this file to choose a Pareto point based on the requirements. The Pareto point that must be implemented can also be specified in the *Co-Flex Toolbox* block in the “Choose_Operation_Point” field as shown in the left snapshot in Figure 4.12. In this snapshot, for example, the first Pareto point is selected.

Parameter writeback: Corresponding to the Pareto point that is selected for the implementation, we can invoke the *Co-Flex: Writeback* tool from the *Co-Flex Toolbox* block to parameterize the Simulink model according. As shown in top left snapshot in Figure 4.13, the control gains are automatically written in the *FeedbackController* block that are then used in the underlying controller implementation. The task schedules are also automatically configured in the corresponding *Dispatch Event Block*s, as shown in the top right snapshot in Figure 4.13. These blocks are then responsible for periodically triggering the tasks. Finally, for the frame schedules, the tool generates a CSV file in a certain format that can be imported directly in the *Database File Block*, as shown in the bottom snapshop in Figure 4.13. Note that the CSV file format is also partially shown in the figure.

Simulation validation: After configuring the software model with the design parameters corresponding to the selected Pareto point, we can verify the control performance of the applications via model-in-the-loop (MIL) simulations. Towards this, we insert the *Simulation Configuration* block from the SIMTOOLS library into the software model. In this block, we select the simulation option “Schedule Timing”, as shown in the top left snapshot in Figure 4.14. Using this option, the ECUs and the communication system are simulated based on the timings of application tasks, communication tasks and bus schedules. We can view the responses of the controlled plants using oscilloscope in Simulink as shown in the top left snapshot in Figure 4.14.

For our case study, we have validated the control performance of the applications for three different Pareto points, i.e., the 1st, the 16th, and the 31st Pareto points. The control responses are recorded for each of the simulation runs and are shown in Figure 4.14. Here, $\mathcal{C}_1$, $\mathcal{C}_3$, $\mathcal{C}_4$, and $\mathcal{C}_5$ are applied unit step references while $\mathcal{C}_2$ moves to the state $x(0) = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T$ on disturbance. We have verified the recorded responses against the expected behavior that we have obtained during controller design and they match perfectly.

It may be observed in Figure 4.14 that for CSS, the three Pareto points select three different sampling period values, i.e., $80\,\mathrm{ms}$, $40\,\mathrm{ms}$, and $5\,\mathrm{ms}$ respectively. Corresponding to these values, the expected settling times of the controllers are $187.3\,\mathrm{ms}$, $101.8\,\mathrm{ms}$, and $63.6\,\mathrm{ms}$ respectively. The settling times of the control responses as obtained via simulations match these expected values. It is evident from the plots in Figure 4.14 that the control response of CSS corresponding to the 16th Pareto point stabilizes faster than in case of the 1st Pareto point. The response for the 31st Pareto point is even faster than that obtained for the 16th Pareto point. For both EWB and CC2, the responses for the 16th and the 31st Pareto points are identical despite different task and message schedules while they differ from the response obtained for the 1st Pareto point. This is because of the fact that the choices of sampling period values for EWB and CC2 respectively remain unchanged for the 16th and the 31st Pareto points while they are different for the 1st Pareto point. However, this verifies the assumption we made for the co-optimization that the control performance depends mainly on the sampling period of the control application.



Simtools Configuration

Simulation Detail Level

( ) Global Timing
( ) Signal Timing
( ) Application Task Timing
[✓ Schedule Timing]

Enable Type Conversion (Physical <--> Coded) ☐

Global Timing Configuration

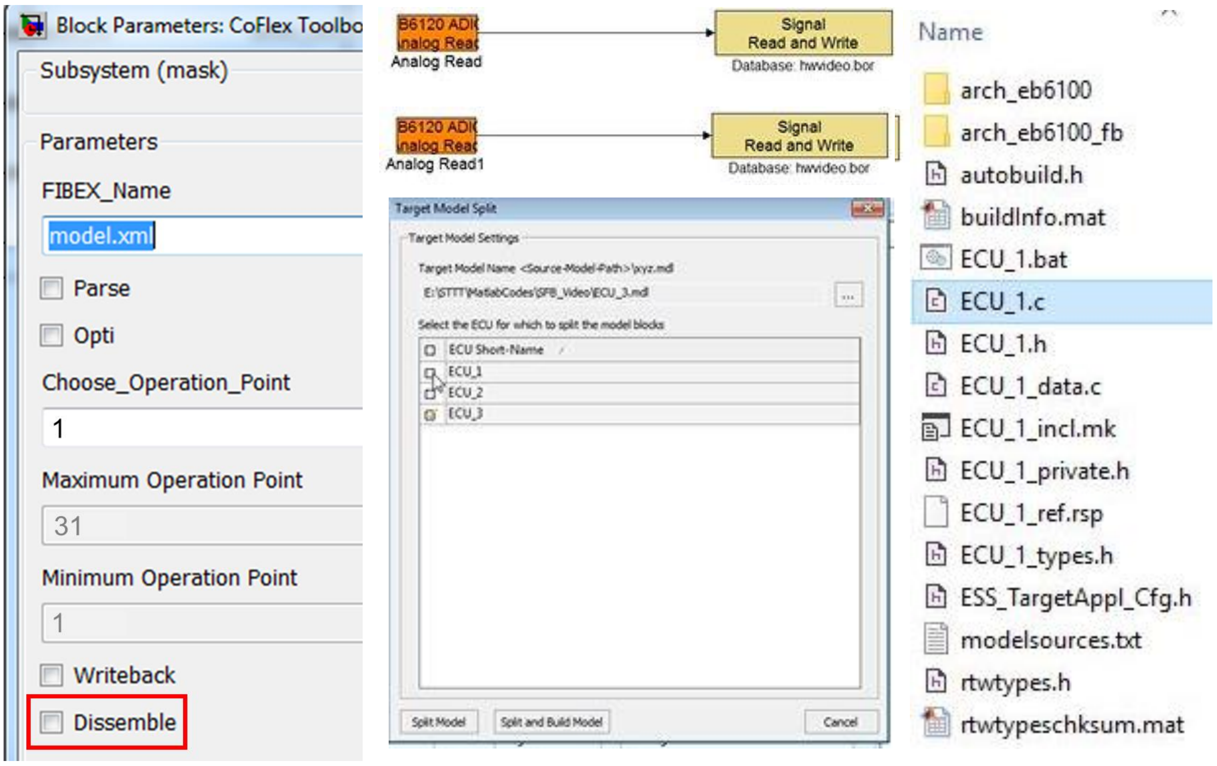
Period [µs]	Offset [µs]	Startup Delay [ms]
20000	0	0

Enable Startup Delay ☐

Figure 4.14: System validation via simulation. (i) SIMTOOLS offers an option to simulate the software functionality based on the timings of the tasks and the messages (top left). (ii) The control responses for the 5 applications are plotted for the 1st, 16th, and the 31st Pareto points (top right, bottom left, and bottom right respectively).

For CC1, it might be noted that despite choosing different sampling periods, i.e., $20\,\mathrm{ms}$ and $5\,\mathrm{ms}$ respectively, for the 16th and the 31st Pareto points, the control responses are similar. This observation also matches the expectation as can be seen in Figure 3.6, where the control performances for the two sampling periods differ by only $6\,\%$ or $20\,\mathrm{ms}$.

Application software modeling and code generation: After validating the synthesized parameters, we invoke the *Co-Flex: Dissemble* tool from the *Co-Flex Toolbox* block by checking the checkbox as shown in the left snapshot in Figure 4.15. This tool automatically generates a new Simulink model from the current model that has all the essential blocks for which codes need to be generated. Note that in the new model, the controller implementation is on the first level and is not masked by any subsystem. This is done to overcome a limitation of the code generation tool, i.e., it does not look under the mask. Furthermore, we manually add certain hardware level details, e.g., to which analog input the sensors are connected to, as shown in the center top snapshot in Figure 4.15. Once all details are added, we use the "Split and Build" option in the *Database File Block*, as shown in the center bottom snapshot in Figure 4.15, to partition the


Block Parameters: CoFlex Toolbo

Subsystem (mask)

Parameters

FIBEX_Name: model.xml

☐ Parse

☐ Opti

Choose_Operation_Point: 1

Maximum Operation Point: 31

Minimum Operation Point: 1

☐ Writeback

☐ Dissemble

Target Model Split

Target Model Settings

Target Model Name <Source-Model-Path>\xyz.mdl

E:\STTT\MatlabCodes\SFB_Video\ECU_3.mdl

Select the ECU for which to split the model blocks

☐	ECU Short-Name
☐	ECU_1
☐	ECU_2
☐	ECU_3

Name

arch_eb6100
arch_eb6100_fb
autobuild.h
buildInfo.mat
ECU_1.bat
ECU_1.c
ECU_1.h
ECU_1_data.c
ECU_1_incl.mk
ECU_1_private.h
ECU_1_ref.rsp
ECU_1_types.h
ESS_TargetAppl_Cfg.h
modelsources.txt
rtwtypes.h
rtwtypeschksum.mat

Figure 4.15: (i) Application software modeling: *Co-Flex: Dissemble* tool (left) automatically generates a new Simulink model for code generation based on the current model by getting rid of the blocks for which codes need not be generated. Hardware-level details, e.g., sensor and actuator ports, are added manually (center top). (ii) Code generation: "Split and Build" option (center bottom) in the *Database File Block* is used to extract the software model for each ECU from which C-codes and the binary file are generated (right).

whole software model into the sub-models corresponding to the ECUs and generate C-code and binaries from these sub-models for the respective ECUs. In our case study, three sub-models are generated for the three ECUs that we have considered. The codes and the binary generated for ECU_1 is shown in the right snapshot in Figure 4.15. *The binary files, thus generated, are then flashed onto ECUs without any error.*

4.5 Related Works

Towards meeting the high performance and low cost requirements for CPSs, it has been emphasized that the co-design of cyber and physical components is the key. That is, the modeling, design, simulation, and validation of computation and communication algorithms, the hardware/software platform, the physical processes, and the human behavior must be considered in an holistic framework [222]. One important problem, in this context, is the co-design of control algorithms and their platform implementations [5]. Although there have been some progress in the state of the art for control-platform co-design, as discussed in Section 3.7, these co-design techniques have not been implemented in practice due to the lack of integrated toolchains.

For this work, firstly, we have reviewed existing industrial toolchains for automotive software development. Major automotive tier 2 suppliers include Vector, dSPACE, Elektrobit, and Siemens Industry Software (formerly known as Mentor Graphics). Vector offers PREEvision [223] where the software architecture and the mapping of software components to ECUs can be specified and the in-vehicle network can be configured. Specification for each ECU can be extracted from PREEvision and imported into DaVinci Configurator Pro, where it is integrated with the basic software and the codes for the software components generated from model-based design tools like Simulink. The developer needs to configure the runtime environment (RTE) for the ECU in the DaVinci Configurator Pro [224] and then the binary file can be generated that will be flashed on the ECU. Similar tool flow is offered by Siemens Industry Software where Capital [225] allows architecture and network design while Volcano Vehicle System Builder [226] enables software integration for the ECUs. In the same vein, dSPACE provides SystemDesk [227] for architecture design and TargetLink [228] for functionality development and code generation. In this work, we have studied software development for FlexRay-based distributed control systems using MATLAB/Simulink and SIMTOOLS/ SIMTARGET (offered by Elektrobit), as discussed in Section 4.2. Note that using the aforementioned toolchains, controllers are designed in model-based design tools while the schedule configuration for ECUs (e.g., in DaVinci Configurator Pro and Volcano Vehicle System Builder) and communication buses (e.g., in PREEvision and Capital) are realized in different tools. None of the toolchains offer co-design of control and platform parameters as we do in the proposed integrated toolchain.

In the context of hardware/software co-design, there also exist certain academic tools like Metropolis [229], Metro II [230], Ptolemy II [231], and Metronomy [232] that allow modeling of heterogeneous components and their co-simulation, thereby enabling design space exploration for heterogeneous systems. Metropolis offers a unified language for capturing different models of computation like dataflow, state machine, and discrete time. Later, Metropolis was extended to Metro II that also allows different specification language for different components. For example, in the case of building design automation, controller can be specified as a Simulink model, specification for the physical building can be in Modelica, while the architecture can be modeled in Metro II [233]. The controller and the building models can be co-simulated with the computing platform in Metro II. Both Metropolis and Metro II use SystemC based engine for the simulation [234]. Ptolemy II can also be used to model functions and architecture in an integrated framework, however, it is more focused on functional modeling allowing several models of computation including process networks, synchronous reactive and continuous time. Metronomy combines the advantages of Ptolemy II and Metro II and allows to model the functions in Ptolemy and the architecture in Metro II. Note that none of the aforementioned tools offer to co-design controllers and their platform implementations, and moreover, they are based on the principle of separation of concerns. Nevertheless, we believe that our proposed co-design technique presented in Chapter 3 can also be integrated in these aforementioned tools. Also, these tools do not have in-built models for the target platform under study and the corresponding basic software. therefore, we have considered a commercial toolchain from Elektrobit that can be used to develop software that will run on a distributed platform comprising EB 6120 ECUs connected over a FlexRay bus.

4.6 Conclusion

In this chapter, we have introduced a step-by-step software development procedure for FlexRay-based distributed control systems that enables correct-by-construction and convenient design and implementation of such systems. We have also developed an integrated toolchain that automates the development process to a large extent. The proposed toolchain integrates a state-of-the-art control-platform co-design approach and the COTS tools for controller design and platform configuration respectively. Therefore, the integrated toolchain reduces the manual efforts on one hand and enables optimal design on the other.

Our proposed toolchain is only a preliminary one showing that such an integrated design and implementation of CPSs is possible in an automotive setting. We would like to extend the toolchain with state-of-the-art techniques considering a more comprehensive design problem including frame packing and task partitioning and mapping. Moreover, the toolchain in its current state only considers LTI feedback control systems. In the future, we can consider nonlinear systems and more complex control techniques like model predictive control, gain scheduling or adaptive control. In our work, we have studied SIMTOOLS and SIMTARGET that are only used for the design of FlexRay-based systems while we would like to investigate the applicability of our proposed design and implementation flow on other communication bus systems like CAN and time-triggered Ethernet. It would be also interesting to study a more complex tool flow for software synthesis of distributed control systems implemented on heterogeneous platforms comprising different communication bus systems and allowing different ECU scheduling policies. We believe that this work would motivate further research towards closing the gap between the state of the art and the state of practice in the context of CPSs design.

5
Tighter Dimensioning of Multi-Resource Cyber-Physical Systems with Control Performance Guarantees[1]

5.1 Introduction

In the last two chapters, we have considered time-triggered implementation of embedded controllers. In such an implementation, software timings are precisely known, and therefore, the controller can be optimally tuned to meet the control requirements. However, the time-triggered implementation is expensive as time-triggered resources may not have a good utilization due to the conservatism involved in their design and allocation. In this chapter, we show that a cost-efficient yet high performance implementation of controllers is possible using a mix of time-triggered and event-triggered resources.

Resource over-dimensioning in safety-critical cyber-physical systems (CPSs): Modern automotive systems deploy a large number of safety-critical functions and many of them are feedback control functions. These functions closely interact with the physical environment using sensors and actuators, and the computation of the control signals is performed on the electrical and electronic (E/E) architecture. Design of such CPSs requires guarantee on functional and timing behavior in all scenarios including the worst case, even if it may rarely occur. This leads to significant resource (computation or communication) over-dimensioning — a particular concern for cost-sensitive domains like the automotive.

[1]This chapter is compiled based on two publications as follows: (i) “Tighter dimensioning of heterogeneous multi-resource autonomous CPS with control performance guarantees” [235] appeared at the 2019 ACM/IEEE Design Automation Conference (DAC). (ii) “GoodSpread: Criticality-aware static scheduling of CPS with multi-QoS resources” [236] appeared at the 2020 IEEE Real-Time Systems Symposium (RTSS).

Interplay between the physical dynamics and the resource requirements: In the context of CPSs, the timing behavior and resource requirements of applications are governed by the dynamics of the physical processes that are being controlled. That is, when the physical plant is in steady state, delay or jitter in closed-loop timings can be tolerated to some extent without jeopardizing safety, however, when there is a disturbance, timing requirements must be strictly satisfied to reject the disturbance quickly. The main message of our work is that by appropriately modeling and analyzing the physical dynamics, a cost-efficient resource allocation is possible. This essentially requires a synergistic study on both platform architecture and physical processes. In particular, in this chapter, we describe a novel interplay between control theory and resource scheduling.

Multi-resource CPS platform under study: Currently, the E/E architecture of automotive systems are highly heterogeneous consisting of different types of resources. High-quality resources often have straightforward timing guarantees, however, they typically have large implementation overheads, and therefore, are expensive. On the other hand, low-quality resources might not offer timing determinism but are significantly cheaper compared to the high-quality resources. In this work, we consider distributed implementation of controllers, where the control signals are exchanged over a FlexRay communication bus. As studied in Section 2.2.2, FlexRay exhibits a hybrid communication protocol offering both TDMA and FTDMA communication. While TDMA is time-deterministic, FTDMA leads to timing jitters. The multi-resource platform under study, therefore, comprises ECUs communicating over a common FlexRay bus. Nevertheless, our proposed implementation can also be applied to other heterogeneous bus systems like TTCAN and TTEthernet or even to a platform offering both wired and wireless communications.

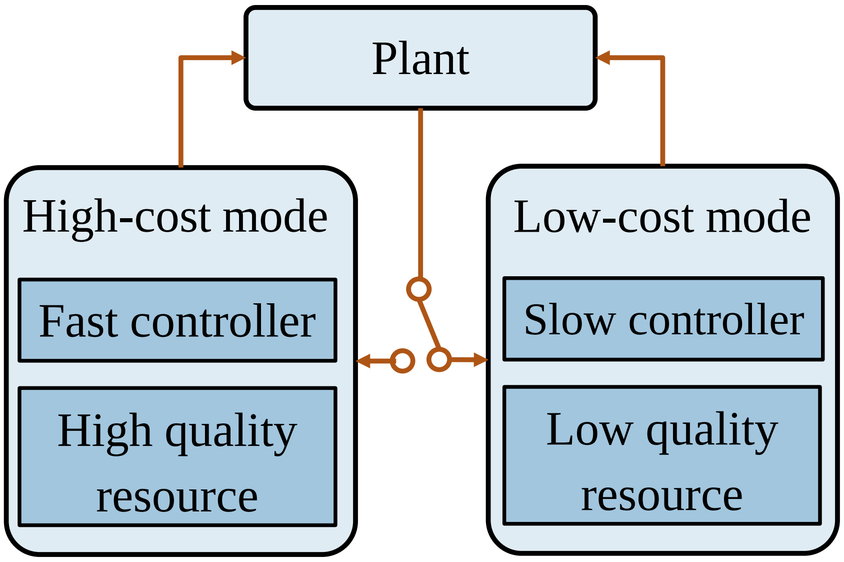


Figure 5.1: Bimodal control strategy over heterogeneous resources. In the high-cost mode, high-quality resources are used by the controller, and therefore, high control performance can be guaranteed. In the low-cost mode, low-quality resources are used by the controller, and therefore, only a lower performance might be obtained.

Bimodal control strategy exploiting heterogeneous resources: In this work, we study a bimodal control strategy comprising a high-cost mode and a low-cost mode, similar to [144], as shown in Figure 5.1. For the multi-resource platform under study, a control signal can be sent using the TDMA (i.e., high-quality) or the FTDMA (i.e., low-quality) communication. In the high-cost mode, the control signal is sent using a TDMA slot and the task schedules can be optimized accordingly to have a negligible delay between the sensing and the actuation. With negligible sensing-to-actuation delay, the controller can be optimized to obtain a high performance. On the other hand, in the low-cost mode, the control signals are sent via the FTDMA communication, and therefore, the communication delay might be non-negligible and variable. Therefore, for the low-cost mode, the controller needs to consider the worst-case, and correspondingly, a slow controller might only be possible that provides a lower control performance. Using a bimodal switching control strategy, it is possible to share TDMA slots among several applications. Here, *the main challenge is to determine how to optimally switch between the two modes such that the usage of TDMA slots is minimized while meeting the performance requirements in all scenarios.*

Contributions: Towards tighter dimensioning of the TDMA slots, we make the following contributions in this chapter:

- Considering the fact that FlexRay, in general, is not dynamically reconfigurable, we propose a strategy to allocate the TDMA slots statically to the applications such that they satisfy their control requirements in all scenarios.
  - Our proposed scheduling strategy first determines for each application, its minimum requirement to stay in the high cost mode. This requirement is derived as the number of times n_T an application must use TDMA slots to send the control data out of n_R consecutive transmissions. Note that this is similar to the weakly hard constraint that is typically expressed in terms of a (m, k)-firm constraint, however, in the scheduling problem under study, the remaining $n_R - n_T$ times the application uses the FTDMA communication to send the control data instead of just dropping them. Note that according to the FlexRay protocol, n_R must be a power of 2, i.e., $n_R = 2^k$, and thus there are limited choices for n_R. We model the closed-loop system comprising the plant and the embedded controller for both high-cost and low-cost modes and analyze the physical dynamics for different combinations of n_T and n_R. Here, we assume that the n_T TDMA slots are allocated consecutively to keep the problem tractable.
  - From the evaluations, we take the value of n_T and n_R that meet the control requirements and results in the minimum usage of the TDMA slots, as the scheduling requirement for the application. Now, with the obtained scheduling requirements for all the applications, we formulate a Satisfiability Modulo Theories (SMT) problem to minimize the total number of slot ids in the FlexRay static segment that must be used for scheduling the applications. The solution to the problem gives a static allocation of TDMA slots to applications in different FlexRay bus cycles.
- Although FlexRay does not allow dynamic reconfiguration, a middleware-based approach is proposed in the literature [237] that allow online slot multiplexing for an ECU. That is, the

middleware can select among several signals in an ECU the one that will be sent on a TDMA slot. Considering such a middleware, we propose a strategy to dynamically allocate TDMA slots to applications as and when it is necessary. Figure 5.2 illustrates the proposed switching control strategy based on dynamic priority assignment that can be described as follows:

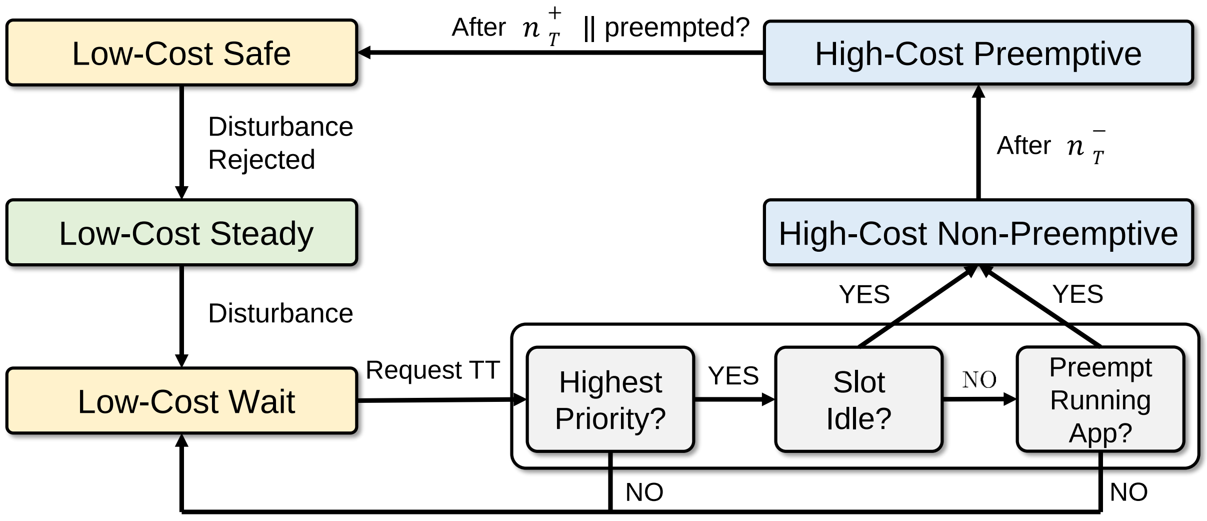


Figure 5.2: The proposed switching control strategy based on dynamic priority assignment. When the closed-loop system is in steady state, the controller is in the low-cost mode. On disturbance, it tries to switch to the high-cost mode based on its availability. It then uses the high-cost mode for a certain minimum number of samples to meet its control requirements.

– According to our proposed strategy, an application stays in the low-cost mode when the controlled plant is in steady state. Here, a stabilizing controller in the low-cost mode will be sufficient to ensure that the plant continues to be in the steady state. Now, when a disturbance arrives, the goal is to reject the disturbance within a certain time. To realize this goal, the application requests the scheduler to switch to the high-cost mode. Depending on the scheduler decision, the application might have to wait before it gets the TDMA slot and switches to the high-cost mode. Corresponding to the number of samples n_w it has waited in the low-cost mode after the disturbance has arrived, there is a minimum number of samples n_T^- that it needs to stay in the high-cost mode to meet the performance requirement. Therefore, once an application switches to the high-cost mode, it stays there non-preemptively for n_T^- samples. Now, depending on n_w, there is also a maximum number of samples till which the application can improve its performance by staying in the high-cost mode. Thus, the application can continue to stay in the high-cost mode till n_T^+ samples if there are no other waiting applications. After n_T^+ samples or if preempted, the application switches back to the low-cost mode where the remaining disturbance (if any) is rejected. Note that the main advantage of the dynamic scheduling over the static scheduling is that the former offers the applications to improve its performance if possible. In static scheduling, the control performance is fixed as additional resources cannot be allocated to an application online even if there are ones not used by applications in their steady states.

  - In this proposed strategy, the scheduler runs a priority-based scheduling scheme, i.e., the application with the highest priority gets the TDMA slot when multiple applications are contesting for it. For each application, we can calculate the maximum number of samples n_w^* for which it cant wait in the low-cost mode without violating the control requirement. A waiting application is dynamically assigned priority based on the maximum number of samples n_w^* it can wait in the low-cost mode. That is, the application for which n_w^* will elapse the soonest will get the highest priority.

- For the proposed dynamic scheduling strategy, we also determine the minimum number of slot ids required in the FlexRay static segment to meet the control requirements of all applications. Here, we assume that each application is mapped to a slot id and one slot id can be shared by several applications. Towards accurate dimensioning of TDMA slots, we propose a nested two-layer technique. In the outer layer, we map applications to slot using a first-fit heuristic. In the inner layer, we validate that the requirements are met for all applications in a given mapping using a formal verification technique. In particular, we model the whole system comprising the applications and the scheduler as a network of timed automata (TAs) and then we use model checking for the verification of TAs models.

  - While there have been previous works on formulating schedulability analysis as model checking problems (e.g., [238, 239]), the main challenge here is to accurately characterize the interplay between the physical dynamics and the scheduling policy. Studying the closed-loop dynamics of the switched system, we can determine the maximum number of samples n_w^* that an application can wait (after a disturbance) for the high-cost mode without violating its performance requirement. Similarly, we can also calculate n_T^- and n_T^+ for all possible values of n_w. Note that n_T^- and n_T^+ are not constants and strongly depend on the controller design and the switching instants. Using the timing variables n_T^-, n_T^+, n_w and n_w^*, it is possible to abstract the behavior of the closed-loop system during a disturbance as a TA. Furthermore, the scheduling policy can also be represented as a TA. Therefore, the whole system can be modeled as a network of TA. We can verify that each of the applications must get the TDMA slots before its n_w^* expires.

- We compare our results with state-of-the-art switching control strategies that assume the TDMA slots to be reconfigurable online and uses a fixed-priority scheduling of the applications onto the TDMA slots. Compared to the state-of-the-art techniques, we can show that for certain cases, our proposed static and dynamic scheduling strategies can save 50% more TDMA slots.

Chapter organization: The rest of this chapter is organized as follows. In the next section, we describe the problem setting. Here, we first study the different types of platform resources in hand and how they impact the control timings. Accordingly, we derive the closed-loop mathematical models for the bimodal controller under consideration. In Section 5.3, we propose a strategy to statically allocate high-quality resources to control applications in a cost-efficient way while taking into consideration the control requirements of the applications. In Section 5.4, we first study the physical dynamics of the controlled plant for the bimodal controller, and accordingly, we derive a cost-efficient switching strategy for the controller. Then, we also outline

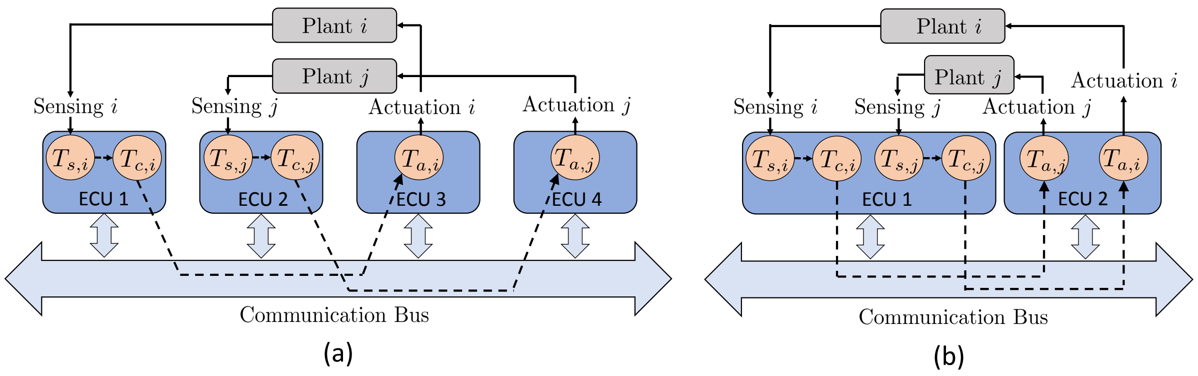


Figure 5.3: Distributed controllers sharing a communication bus. Two different cases are shown: (a) The ECUs sending and receiving the control data respectively are different for different applications. In this case, it is only possible to statically map control data to TDMA communication slots. (b) Control data are sent between the same pair of ECUs. In this case, middleware-based online reconfiguration of TDMA slots is possible.

a scheduler that will dynamically allocate high-quality resources to applications. Section 5.5 describes the proposed nested two layer technique to accurately dimension the high-quality resources for the proposed dynamic scheduling strategy. In Section 5.6, we consider a case study to which we apply the proposed scheduling strategies (static and dynamic) and outline the results. We discuss the related works in Section 5.7 and provide the concluding remarks in Section 5.8.

5.2 Problem Setting

We consider a distributed automotive setting where several ECUs communicate over a shared communication bus, as shown in Figure 5.3. Multiple control applications, denoted by $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, are implemented on this distributed E/E platform. Each application $\mathcal{C}_i$ is implemented using three tasks $(T_{s,i}, T_{c,i}, T_{a,i})$ that is similar to the problem setting described in Section 3.2.1, where $T_{s,i}$ is the sensor task, $T_{c,i}$ is the controller task, and $T_{a,i}$ is the actuator task. $T_{s,i}$ and $T_{c,i}$ are mapped on the same ECU while $T_{a,i}$ is mapped on a different ECU. The control input is calculated by $T_{c,i}$ and is then communicated to $T_{a,i}$ over the shared bus. This is a common scenario in the automotive setting due to the spatial distribution of sensors and actuators. Note that the analysis presented in this chapter can also be applied to other settings with different task partitioning and mapping schemes.

5.2.1 Heterogeneous Resources

In this work, we consider the provision of both high-quality and low-quality resources for the implementation of a controller. In this context, we study the FlexRay communication bus. As described in Section 2.2.2, FlexRay is a hybrid communication protocol where each bus cycle is composed of a static segment and a dynamic segment. The static segment exhibits time-deterministic communication and comprises a number of TDMA slots of equal length. Therefore, a message assigned to a static slot is transmitted within the corresponding time window. The start and the end of a message transmission are precisely known. Therefore, TDMA slots in the static segment are high-quality resources. The dynamic segment implements FTDMA communication and is partitioned into a number of minislots of equal length, where a minislot is much shorter than a slot in the static segment. A message assigned to the dynamic segment may consume more than one minislot. Thus, the timing of a message depends on other preceding messages. This results in time-varying transmission delay while the worst case may still be determined [107]. The timing jitters in the dynamic segment render the communication low-quality.

FlexRay schedules are typically static and configured offline. In Section 2.2.2, we define a FlexRay message schedule as a tuple of slot id, base cycle and repetition rate. That is, the schedule of a control message $m_{c,i}$ corresponding to the application $\mathcal{C}_i$ is represented using a tuple $\{s_{c,i}, b_{c,i}, r_{c,i}\}$ where $s_{c,i}$, $b_{c,i}$, and $r_{c,i}$ represent the slot id, the base cycle, and the repetition rate respectively. While configuring the FlexRay schedules, we must consider the constraint that no two messages are assigned the same slot in a FlexRay cycle. However, one slot can be provisioned for different messages in different cycles. This enables sharing of TDMA slots in FlexRay that we exploit in this work. Let us consider two application $\mathcal{C}_i$ and $\mathcal{C}_j$ that are implemented as shown in Figure 5.3. Both applications send the control message over the FlexRay bus. Let us consider the schedules for $m_{c,i}$ and $m_{c,j}$ as $\{5, 0, 2\}$ and $\{5, 1, 2\}$ respectively. Here, both messages share slot 5, i.e., $m_{c,i}$ is sent on the slot in the even cycles while $m_{c,j}$ is sent on the slot in the odd cycles. Also note that in this case, the schedules are static, i.e., $m_{c,i}$ cannot be sent on the slot in the odd cycles even if it improves the performance of $\mathcal{C}_i$ while there is no disturbance experienced by $\mathcal{C}_j$.

In this chapter, we also consider dynamic scheduling of messages on TDMA slots. However, FlexRay schedules, as per the definition, are not runtime-configurable. Here, we exploit the middleware-based solution proposed in [237] that enable dynamic scheduling of FlexRay messages with certain restriction. This middleware can be explained using the example in Figure 5.3(b). Here, ECU 1 needs to send two control data to ECU 2. However, it is desirable to send the high-priority data using a TDMA slot while the low-priority data can be sent on the dynamic segment. Here, the priorities can change during runtime. Now, let us consider that there is a middleware task T_{MW} in ECU 1 that sends two messages m_{st} and m_{dyn} to ECU 2 in each FlexRay cycle. Here, m_{st} is scheduled using a TDMA slot in the static segment while m_{dyn} is sent on the dynamic segment. T_{MW} reads both control data, and based on their priorities, it decides in each FlexRay cycle, the mapping of data to the messages m_{st} and m_{dyn}. Thus, effectively, in each cycle, only one of the control data is sent on the high-quality TDMA slot while the other one is sent on the dynamic segment. Here, T_{MW} does not only send the control data but also an identifier that determines the sending task. Correspondingly, on the receiving side, each of the tasks $T_{a,i}$ and $T_{a,j}$ will read the messages m_{st} and m_{dyn} and check the respective identifier to determine if the control data in the message is useful to the task, and act accordingly. The middleware task, here, enables sharing of a TDMA slot among several applications, where the control data can be scheduled dynamically on the slot. However, note that this solution imposes a restriction on the mapping of data to a slot. That is, a slot can be used to transmit data from one sender ECU only.

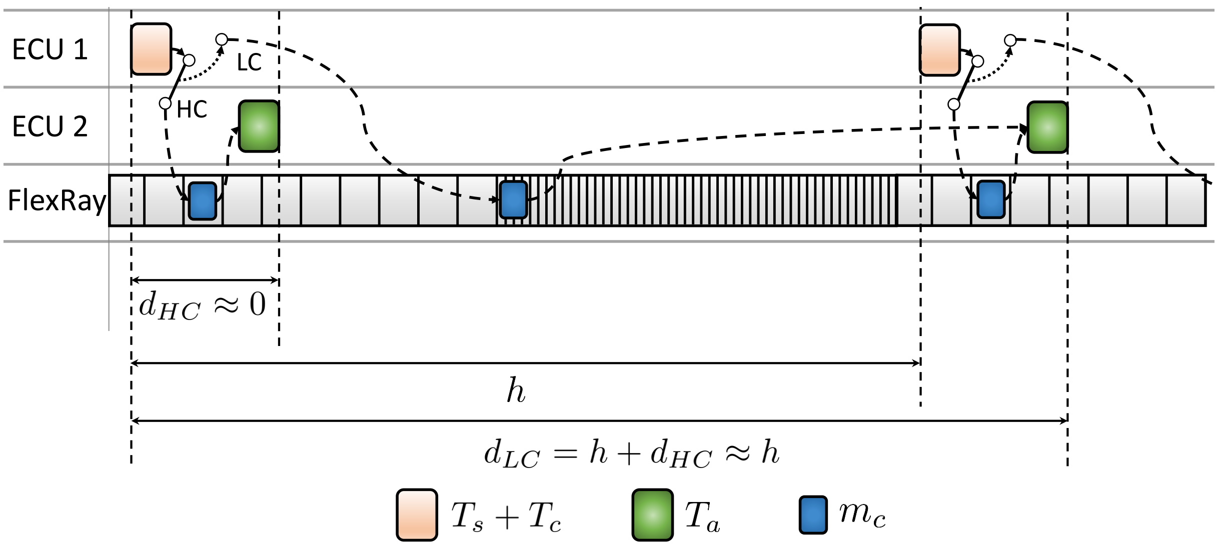


Figure 5.4: Closed-loop timings for the bi-modal controller. The control input generated by the controller task T_c can be sent either on a TDMA slot in the static segment (if available) or over the dynamic segment. When the control input is sent on the static segment, the sensing-to-actuation delay d_{HC} can be optimized to a negligible value compared to the sampling period h. When the control input is sent on the dynamic segment, the sensing-to-actuation delay d_{LC} becomes approximately equal to the sampling period h.

5.2.2 Bimodal Control Strategy

We consider linear and time-invariant (LTI) physical plants for which the continuous-time mathematical model is given by Eq. (2.1). In this work, the controller is implemented using periodic software tasks that are mapped in a distributed manner. The sensor and the controller tasks (i.e., T_s and T_c respectively) are mapped on one ECU while the actuator task (i.e., T_a) is mapped on a different ECU. In such a distributed setting, control data is sent over the communication bus, i.e., FlexRay. Here, the sampling period, given by the time interval between two consecutive sensing operations, is a constant h, as shown in Figure 5.4.

In the bimodal control strategy under study, the controller in the high-cost mode uses the TDMA slots in the static segment of FlexRay for the communication of the control input. For such a time-triggered communication, the timings of a message transmission is exactly known. This allows to optimally schedule the tasks such that the delay d_{HC} between the sensing and the actuation operations is negligible compared to the sampling period. Here, the execution times of tasks and the length of TDMA slots adds up to a few hundred microseconds, while the sampling period is in the range of tens of milliseconds. Therefore, we can assume $d_{HC} \approx 0$, as shown in Figure 5.4. The feedback controller for the high-cost mode can be designed as follows:

$$u[k] = -K_{HC}x[k], \tag{5.1}$$

where K_{HC} is the feedback gain. Combining the discrete-time equivalent model of the plant in Eq. (2.5) and the controller model in Eq. (5.1), the closed-loop system dynamics is given by:

$$x[k+1] = (\phi - \Gamma K_{HC})x[k]. \tag{5.2}$$

Eigenvalues of the closed-loop system matrix $(\phi - \Gamma K_{HC})$ determine the control performance, such as settling time. Here, settling time is defined as the period of time the controller takes to reject the disturbance and get the plant back to the steady state. Optimization-driven pole-placement controller design technique can be used similar to Algorithm 2.

On the other hand, in the low-cost mode, control data is transmitted in the dynamic segment and for the task schedules, as shown in Figure 5.4, the sensing-to-actuation delay becomes approximately equal to the sampling period, i.e., $d_{LC} \approx h$. Note that the control data can be sent anywhere in the dynamic segment depending on its schedule and the corresponding preceding messages. However, we assume that the message will not be dropped as can be verified from the latest finish time of the transmission in the dynamic segment as per Eq. (2.42). For one sample delay, as is the case in the low-cost mode, at $t[k]$, $u[k-1]$ is applied to the plant and is held till $t[k+1]$. Thus, the plant model can be derived from Eq. (2.14) as follows:

$$x[k+1] = \phi x[k] + \Gamma_1 u[k-1], \quad y[k] = Cx[k]. \tag{5.3}$$

For this delayed system, an augmented state vector $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$ is considered, as discussed in Section 2.1.3. Therefore, the stabilizing control law becomes:

$$u[k] = -K_{LC} z[k] = -K_{LC} \begin{bmatrix} x[k] \\ u[k-1] \end{bmatrix}. \tag{5.4}$$

For the system with augmented state z[k], pole-placement can be applied to design K_{LC} as discussed in Section 2.1.5.

Switching stability: Using the bimodal controller, we switch between two closed-loop dynamics corresponding to the high-cost and the low-cost modes respectively. Typically, to ensure that such a switched system will be stable for any arbitrary switching between the two modes, we need to verify for switching stability in addition to designing stable controllers (K_{HC} and K_{LC}) separately for the individual modes. A survey on stability of switched linear systems can be found in [240].

Note that the controllers K_{HC} and K_{LC} are of different dimensions. Thus, we write the discrete-time state space model in the high-cost mode for the augmented states z[k] as follows:

$$z[k+1] = \begin{bmatrix} \phi & 0 \\ 0 & 0 \end{bmatrix} z[k] + \begin{bmatrix} \Gamma \\ 1 \end{bmatrix} u[k] = \left(\begin{bmatrix} \phi & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} \Gamma \\ 1 \end{bmatrix} \begin{bmatrix} K_{HC} & 0 \end{bmatrix} \right) z[k] = \Phi_{HC} z[k]. \tag{5.5}$$

On the other hand, the state-space model representing the low-cost mode for the augmented states can be written as follows:

$$z[k+1] = \begin{bmatrix} \phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} z[k] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u[k] = \left(\begin{bmatrix} \phi & \Gamma_1 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} K_{LC} \right) z[k] = \Phi_{LC} z[k]. \tag{5.6}$$

To ensure switching stability between the two closed-loop subsystems, given by Eq. (5.5) and Eq. (5.6) respectively, there must exist a positive definite matrix $\mathbf{P}$ such that:

$$\begin{aligned} \Phi_{HC}^T \cdot \mathbf{P} \cdot \Phi_{HC} - \mathbf{P} < 0 \\ \Phi_{LC}^T \cdot \mathbf{P} \cdot \Phi_{LC} - \mathbf{P} < 0. \end{aligned} \tag{5.7}$$

Here, $z[k]^T \cdot \mathbf{P} \cdot z[k]$ is a common quadratic Lyapunov function (CQLF) and its value decreases with every discrete time step even during switching from one subsystem to another.

Note that our proposed static and dynamic scheduling strategies are oblivious of the existence of a CQLF for the switched system. For a given pair of controllers, K_{HC} and K_{LC}, we study the control dynamics for all switching sequences corresponding to each of the prospective schedules. In our proposed static scheduling scheme, we select one of the prospective schedules such that it allows only those switching sequences that meet the control requirements. That is, the synthesized static schedule will be correct-by-construction. On the other hand, for the proposed dynamic scheduling strategy, we verify that for all switching sequences as allowed by the scheduler, the control requirements must be met. Thus, our proposed scheduling strategies can meet the requirements even in those cases where the high-cost and the low-cost modes do not satisfy the switching stability condition given in Eq. (5.7).

However, intuitively, if switching between the two subsystems are not stable, there might be an increase in the system's energy during switching, and therefore, the controller might require more time to stabilize the system. Alternatively, this would imply that the controller would need to stay in the high-cost mode for a longer time to meet the control requirements, and therefore, would require more high-quality resources. As the main goal of this work is to minimize the usage of high-quality resources, we recommend that the two controllers, i.e., K_{HC} and K_{LC}, are designed such that they satisfy the switching stability condition given in Eq. (5.7). Later, in Section 5.3.2 and Section 5.4.3, using a motivational example, we evaluate the impact of the existence of a CQLF on the performance of the switching control strategy under study.

5.3 Statically Scheduling the Bimodal Controllers

Typically, safety-critical control applications need to meet stringent performance requirements. Thus, they are implemented using only high-quality resources to extract maximum control performance. However, high-quality resources are also expensive and they are limited in quantity. That is, these resources easily become scarce when increasingly more application are implemented using them. Thus, the main idea in this chapter is to share these resources among multiple safety-critical applications without jeopardizing the control requirements. As discussed in Section 5.2.2, the bimodal controller that we consider here, can either stay in the high-cost mode where it uses the high-quality resources or it can move to the low-cost mode and use low-quality resources. In this section, for the problem setting under study, we propose a novel strategy to statically allocate the high-quality resources (i.e., TDMA slots in the FlexRay static segment) to the bimodal controllers such that each of them meet their respective control requirements. Our proposed strategy comprises two stages: (i) In the first stage, we derive, for each application, the minimum amount of high-quality resources required to meet the control requirements. (ii) In the second stage, we formulate an optimization problem, to statically allocate the high-quality resources to applications so that the resource requirements are met. The rest of this section describes our proposed strategy in greater detail.

5.3.1 Stage 1: Deriving Minimum Resource Requirements

As mentioned before, the TDMA slots in the static segment of FlexRay communication cycle is regarded as high-quality resources. Typically, when a control application uses only these TDMA slots for communication, then one slot is assigned periodically to the application based on the period with which the controlled plant is sampled. Let the sampling period for an application $\mathcal{C}_i$ be h_i. For a bus cycle time T_{bus}, the control data is sent as a message $m_{c,i}$ for which the repetition rate is $r_{c,i} = \frac{h_i}{T_{bus}}$. Now, if we assume that the number of configurable communication cycles for FlexRay be N_{com}, then the total number of slots used by the message in every N_{com} cycles is equal to $\frac{N_{com}}{r_{c,i}}$. However, for the bimodal controller under study, it might not be required to provide a TDMA slot for every message instance as the message can alternatively be sent on the dynamic segment. Here, we suggest a scalable technique to derive the number of slots that must be assigned to an application for sending the control data in N_{com} communication cycles such that the control requirements are met in all scenarios. Note that according to the FlexRay protocol, the slot assignments repeat every N_{com} cycles.

In this work, we consider the settling time J_i as the performance metric which is defined as the time taken by a controller to bring the system back to a steady state after a disturbance. We denote J_i^* as the minimum settling time requirement. We further consider the standard sporadic model for disturbance arrivals, i.e., two disturbances are separated by a time $t_{a,i}$ where $J_i^* < t_{a,i}$. Here, a new disturbance will not arrive until the preceding one is completely rejected. If we guarantee that the designed controller will stabilize the system before J_i^*, then we ensure that two successive disturbances will never overlap.

Now, let $J_{HC,i}$ be the settling time of the system when the controller always stays in the high-cost mode, i.e., $K_{HC,i}$ is applied and the static segment is used for the transmission of control data. We further denote $J_{LC,i}$ as the settling time of the system when the low-cost mode of the controller is exclusively used, i.e., $K_{LC,i}$ is applied and the control data is sent on the dynamic segment. Now, in this chapter, we are interested in the case where $J_{HC,i} < J_i^* < J_{LC,i}$. In such a case, on one hand, if the controller always stays in the low-cost mode then the control requirements will not be met, and therefore, the controller needs to switch to the high-cost mode. On the other hand, the controller might not be required to stay in the high-cost mode exclusively and switching back to the low-cost mode for a certain number of samples will not jeopardize the control requirements. Now, the primary question is: *how the controller must switch between the two modes such that the requirements are met in all possible scenarios?*

It is challenging to deduce a mathematical expression for the settling time of a system in terms of the fraction of time the controller stays in the high-cost mode. Moreover, the settling time will depend on the exact sequence of the control modes after a disturbance. Let us consider a case where the sampling period of an application $\mathcal{C}_i$ is equal to the bus cycle time, i.e., $h_i = T_{bus}$. For $N_{com} = 64$ cycles, let us consider that the controller sends the control data on the static segment in 32 cycles while for the remaining cycles it uses the dynamic segment for communication. In such a case, there are more than 10^{18} possible slot allocations in the static segment for the application. To guarantee that the settling time requirements will be met for any such slot allocation, we have to perform closed-loop simulations for all such cases. However, this is not feasible in practice. Therefore, towards a more scalable approach, we only consider schedules where TDMA slots are allocated for $n_{T,i}$ consecutive transmissions of control data within $n_{R,i}$ transmissions. Here, $n_{R,i}$ can take values in $\{2^k | 0 \le k \le \log_2 \frac{N_{com}}{r_{c,i}}\}$. The value of $n_{R,i}$ is constrained due to the fact that FlexRay schedules repeat every N_{com} cycles and $N_{com} = 2^k$, where $k \in \mathbb{W}$. Now, for the case where $n_{R,i} = 64$ and $n_{T,i} = 32$, there are only 64 possible slot allocations that we need to evaluate. This can be done in reasonable time.

Algorithm 5: Deriving the resource requirements for a control application.

```
Parameter   : $T_{bus}$, $N_{com}$
Input       : $(A_i, B_i, \mathcal{C}_i)$, $h_i$, $K_{HC,i}$, $K_{LC,i}$, $J_i^*$
Output      : $n_{R,i}$, $n_{T,i}$
1  $\overline{n}_R = \frac{N_{com} \cdot T_{bus}}{h_i}$;
2  $n_{R,i} = 1$;
3  $n_{T,i} = 1$;
4  for $k \leftarrow 1$ to $\log_2 \overline{n}_R$ do
5      $n_R^k = 2^k$;
6      $LT = 0$;
7      $RT = n_R^k$;
8      for $q \leftarrow 1$ to $\log_2 n_R^k$ do
9          $n_T^k = \frac{LT+RT}{2}$;
10         $SEQ = \left[\textbf{Zeros}(1, n_R^k - n_T^q) \quad \textbf{Ones}(1, n_T^q)\right]$;
11         for $m \leftarrow 1$ to $n_R^k$ do
12             $J_i^m =$ Simulate $((A_i, B_i, \mathcal{C}_i), h_i, K_{HC,i}, K_{LC,i}, SEQ)$;
13             if $J_i^m > J_i^*$ then
14                 $LT = n_T^k$;
15                 break;
16             else if $m == n_R^k$ then
17                 $RT = n_T^k$;
18             else
19                 $SEQ = SEQ \ll 1$;
20             end
21         end
22     end
23     $n_T^k = RT$;
24     if $\frac{n_T^k}{n_R^k} < \frac{n_{T,i}}{n_{R,i}}$ then
25         $n_{T,i} = n_T^k$;
26         $n_{R,i} = n_R^k$;
27     end
28 end
29 return $n_{R,i}, n_{T,i}$;
```

Algorithm 5: We employ this algorithm to derive the minimum resource requirements, i.e., in terms of $n_{T,i}$ and $n_{R,i}$, for a control application $\mathcal{C}_i$. The parameters of the algorithm are the given values of the bus cycle time T_{bus} and the number of configurable FlexRay communication

cycles N_{com} respectively. It takes as input the continuous-time plant model (A_i, B_i, C_i), the sampling period h_i, the control gains, $K_{HC,i}$ and $K_{LC,i}$, for the high-cost and the low-cost modes respectively, and the settling time requirement J_i^*. It generates as output the minimum required high-quality resources in terms of the number of consecutive control instances $n_{T,i}$ for which the controller must stay in the high-cost mode out of $n_{R,i}$ instances, where the controller switches to the low-cost mode for the remaining instances.

In line 1 of the algorithm, we calculate the number of control instances $\overline{n}_R$ in N_{com} consecutive communication cycles for the application. This also corresponds to the number of TDMA slots that can be maximally assigned to the application within N_{com} communication cycles. In lines 2 and 3, we initialize $n_{R,i}$ and $n_{T,i}$ respectively as 1. Note that this initialization corresponds to the maximum resource requirement, i.e., in each control instance the controller operates in the high-cost mode or the control data is sent exclusively on the static segment.

As mentioned earlier, $n_{R,i}$ can take only discrete number of values constrained by the FlexRay protocol. As assigned in lines 2 and 3, for $n_{R,i} = 1$, $n_{T,i} = 1$, i.e., the controller operates only in the high-cost mode. This is because of the fact that the only other possible assignment is $n_{T,i} = 0$, in which case, the controller is always in the low-cost mode, and therefore, the performance requirement is violated as $J_{LC,i} > J_i^*$. Now, the for loop in lines 4-26 iterates through the remaining possible values of $n_{R,i}$ given by $\{2^k | 1 \leq k \leq \log_2 \overline{n}_R\}$. In line 5, we determine the choice of $n_{R,i}$ in the current iteration as n_R^k based on the loop variable k. Corresponding to n_R^k, we use a binary search algorithm to determine the number of consecutive control instances n_T^k for which the controller must stay in the high-cost mode (lines 6-21).

Note that n_T^k will have a value between 0 and n_R^k. Correspondingly, we initialize the lower and upper limits (LT and RT) of the search as 0 and n_R^k respectively in lines 6 and 7. Using this technique, it will take $log_2 n_R^k$ iterations to determine the value of n_T^k (lines 8-20). In each iteration, we take the midpoint between the lower and the upper limits as n_T^k (line 9). Based on the values of n_T^k and n_R^k, we formulate an initial sequence (SEQ) of control modes where $n_R^k - n_T^k$ instances in low-cost mode is followed by n_T^k instances in high-cost mode (line 10). For example, if $n_T^k = 3$ and $n_R^k = 8$, then SEQ is initialized as 00000111, where 0 stands for the low-cost mode and 1 represents the high-cost mode. Note that for $0 < n_T^k < n_R^k$, there are exactly n_R^k possible sequences, where the controller stays in the high-cost mode for n_T^k consecutive instances before switching to the low-cost mode. Furthermore, note that a sequence repeats infinitely, and therefore, for both sequences, $SEQ = 10000011$ and $SEQ = 00000111$, the controller stay in the high-cost mode for three consecutive control instances before switching to the low-cost mode for 5 consecutive instances. Moreover, for any periodic sequence of n_R^k instances, there are exactly n_R^k distinct points where a disturbance can arrive, and it is necessary to evaluate the control response for all possible scenarios. Thus, for the current choice of n_R^k and n_T^k, we evaluate all possible sequences in lines 11-19.

For each sequence, we simulate the closed-loop system based on the continuous-time plant model (A_i, B_i, C_i), the sampling period h_i, and the control gains in the two modes ($K_{HC,i}$ and $K_{LC,i}$). From the simulated control response, we calculate the value of the settling time J_i^m (line 12). If the obtained settling time violates the minimum requirement then we update the lower limit as the current value of n_R^T and do not evaluate any further sequence for this value of n_R^T (lines 13-15). Here, the assumption is that for an increasing number of instances in the high-cost mode the settling time improves. Thus, by updating the lower limit, we notify the algorithm to only look for higher number of instances in the high-cost mode in the next iterations as the current number of instances is not sufficient. Moreover, as for the current value of n_T^k, there exists a certain disturbance arrival point for which the settling time requirement is not met, we cannot take this value as feasible, and therefore, there is no need to evaluate further sequences for the same value of n_T^k. On the other hand, if the obtained settling time meets the requirement, then we further check if we have evaluated all possible sequences for the current value of n_T^k (line 16). If this is the case, then it implies that the value of n_T^k is feasible and we update the upper limit with the current value of n_T^k (line 17). This is done to notify the algorithm that we can now look if there exists a lower value of n_T^k that meets the requirement. Note that the goal is to determine the minimum requirement for the high-quality resources. Now, if all sequences are not evaluated for the current value of n_T^k, then the next sequence is obtained by shifting the literals in SEQ to the left by one position (lines 18-20). Note that this is a circular left shift, e.g., $11100000 \ll 1 = 11000001$.

Note that the current value of the search's upper limit gives the minimum value of n_T^k that satisfies the requirement (line 23). Now, we compare if the feasible combination of n_R^k and n_T^k obtained in the current iteration will take less slots on average than the best solution obtained so far (line 24-27). If this is the case, then we update the best solution in lines 25 and 26. In line 29, the algorithm returns the minimum resource requirements for the application in terms of $n_{R,i}$ and $n_{T,i}$.

Summary: We can use Algorithm 5 to determine the minimum resource requirement for each application. The resource requirement is expressed in terms of the number of control instances $n_{T,i}$ for which the controller must consecutively stay in the high-cost mode out of $n_{R,i}$ instances. This information can be used to formulate an optimization problem for slot allocation. Also, note that using this algorithm, we do not guarantee optimality. This is because we do not evaluate several sequences of control modes to preserve the scalability of our approach.

5.3.2 A Motivational Example

A DC motor position control system [241] is considered for which the continuous-time plant model is given by:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -0.0227 & 54.5455 \\ 0 & -34.2857 & -70 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 28.1754 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}. \tag{5.8}$$

It is desired to keep the position at $y = 0$ and on disturbance the system moves to a state given by $x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. The settling time threshold is assumed as $||y[k]|| \leq 0.02$, $\forall k \geq J$. The settling time requirement is $J \leq 0.36\,\text{s}$.

For a sampling period of $h = 0.02\,\text{s}$, we synthesize a control gain K_{HC} for the high-cost mode and two control gains, K_{LC}^s and K_{LC}^u, for the low-cost mode. They are given as follows:

$$K_{HC} = \begin{bmatrix} 30 & 1.2626 & 1.1071 \end{bmatrix}, \tag{5.9}$$

$$K_{LC}^s = \begin{bmatrix} 13.8921 & 0.5773 & 0.8672 & 1.0866 \end{bmatrix}, \tag{5.10}$$

$$K_{LC}^u = \begin{bmatrix} 2.9120 & -0.6141 & -1.0399 & 0.1741 \end{bmatrix}. \tag{5.11}$$

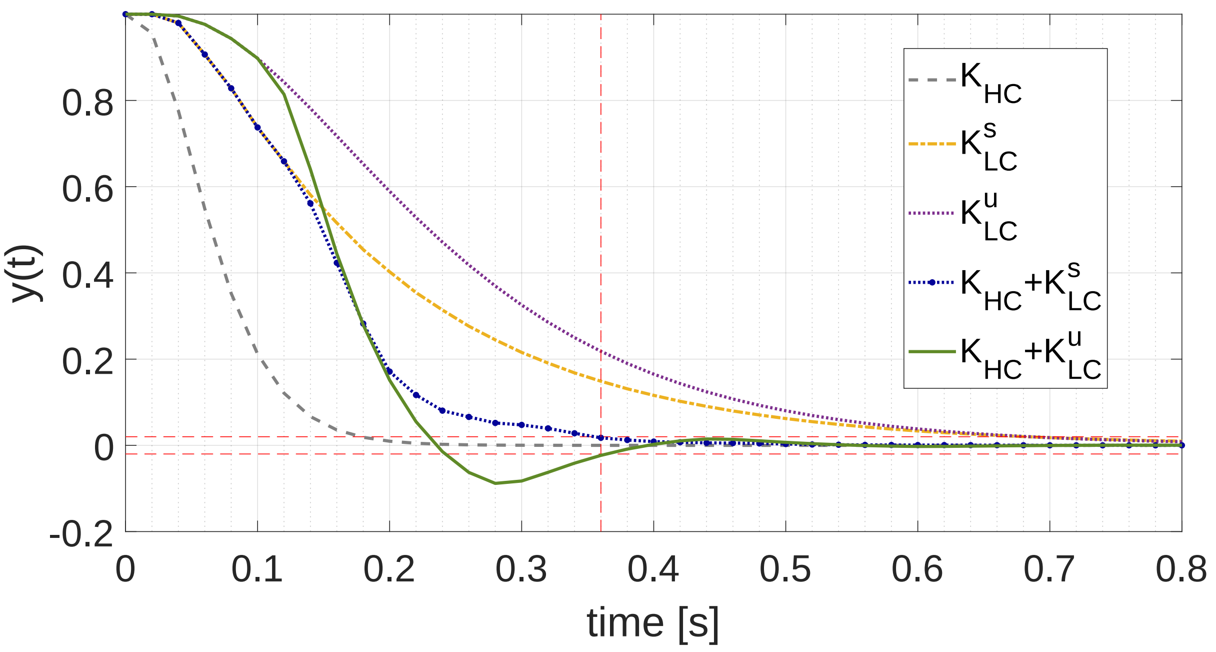


Figure 5.5: Control responses for different static schedules. The gray dashed line shows the response when the controller only operates in the high-cost mode and uses K_{HC}. The orange dash-dotted line and the purple dotted line show the responses when the controller operates only in the low-cost mode using the control gains K^s_{LC} and K^u_{LC} respectively. The blue marked-dotted line and the green solid line show the control responses when bimodal controllers, i.e., $\{K_{HC}, K^s_{LC}\}$ and $\{K_{HC}, K^u_{LC}\}$ respectively, are used.

When the controller uses K_{HC} in the high-cost mode and K^s_{LC} in the low-cost mode, switching between the modes is stable, i.e., there exists a CQLF as per Eq. (5.7). However, arbitrary switching between the two modes might not be stable when K^u_{LC} and K_{HC} are used in the low-cost and the high-cost modes respectively.

In Figure 5.5, we show control responses for different static schedules. When the controller only operates in the high-cost mode using the control gain K_{HC}, the settling time is $0.18\,\text{s}$ (gray dashed line). When the controller operates only in the low-cost mode using the control gains K^s_{LC} and K^u_{LC} respectively, the settling time becomes $0.7\,\text{s}$ (orange dash-dotted and purple dotted line respectively). When the controller stays in the low-cost mode and uses K^s_{LC} for six control instances followed by two instances in the high-cost mode using K_{HC}, and then repeats, the settling time is $0.36\,\text{s}$. Here, the controller uses the high-quality resources 25% of the time. On the other hand, when the controller stay four instances in the low-cost mode followed by four instances in the high-cost mode and then repeats, where it uses K_{HC} in the high-cost (HC) mode and K^u_{LC} in the low-cost (LC) mode, the settling time is also $0.36\,\text{s}$. In this case, the high-quality resources are used 50% of the time.

There are two important observations here: (i) By switching to the high cost mode for 25% of the instances, we can improve the settling time by almost 50%. Thus, using our proposed strategy, it is possible to save high-quality resources significantly. (ii) When the bimodal controller is designed satisfying the switching stability condition, we can save more resources (i.e.,

Table 5.1: Deriving resource requirements for the DC motor position control application

		No. of instances in the repeating sequence (n_R^k)				
		1	2	4	8	16
No. of consecutive instances in the high-cost mode (n_T^k)	$K_{HC} + K_{LC}^s$	1	1	1	2	4
	$K_{HC} + K_{LC}^u$	1	1	2	4	8

25% high-quality resources) compared to the case when switching stability condition is ignored. Thus, we suggest to design the controllers for the low-cost and the high-cost modes respectively such that there exists a CQLF for the closed-loop dynamics representing the two modes. This will ensure that during switching between modes the energy of the system will always decrease.

Furthermore, we apply Algorithm 5 to determine the resource requirements for this control application. Let us assume a bus cycle time $T_{bus} = 0.005\,\text{s}$ and $N_{com} = 64$. Now, for a sampling period $h = 0.02\,\text{s}$, n_R can take values in $\{1, 2, 4, 8, 16\}$. Corresponding to each $n_R^k \in \{1, 2, 4, 8, 16\}$, we compute the value of n_T^k for both bimodal controller, i.e., $\{K_{HC}, K_{LC}^s\}$ and $\{K_{HC}, K_{LC}^u\}$ respectively, and we tabulate the results in Table 5.1. It can be observed that when K_{HC} and K_{LC}^s are used in the high-cost and the low-cost modes respectively, allocating the high-quality resources for 25% of the control instances is enough to meet the performance requirements for $n_R^k \in \{4, 8, 16\}$. However, when K_{LC}^u is used in the low-cost mode, the controller needs the high-quality resources in 50% of the instances for $n_R^k \in \{2, 4, 8, 16\}$. This again shows that considering switching stability during controller design can reduce usage of high-quality resources. Using Algorithm 5, we can derive the resource requirement as $n_T = 1$ and $n_R = 4$ for the bimodal controller using K_{HC} in the high-cost mode and K_{LC}^s in the low-cost mode. That is, out of four consecutive control instances, exactly in one instance, high-quality TDMA slot is used to send the control input.

5.3.3 Stage 2: Optimal Allocation of the TDMA Slots

In Stage 1, based on the control requirements, we determine the minimum resource requirements for each application. In this stage, we take the resource requirements as input and determine a feasible static allocation of slots to applications such that the requirements are met. As discussed in Section 5.2.1, a FlexRay schedule is denoted using a tuple of slot id, base cycle, and repetition rate. In the slot allocation problem that we are considering here, it is desirable to minimize the number of slot ids assigned to the bimodal controllers. This essentially improves the extensibility of FlexRay schedules.

Let us consider two bimodal controllers ($\mathcal{C}_1$ and $\mathcal{C}_2$), as described in Section 5.2.2, where each has a sampling period that is equal to the bus cycle time, i.e., $h_1 = h_2 = T_{bus}$. Now, both need to spend $n_{T,1} = n_{T,2} = 4$ consecutive control instances in the high-cost mode out of $n_{R,1} = n_{R,2} = 8$ instances. Let us consider two different schedules for this example.

- *Case 1*: The control data for $\mathcal{C}_1$ is sent using four messages that are scheduled as $\{3, 0, 8\}$, $\{3, 1, 8\}$, $\{3, 2, 8\}$, and $\{3, 3, 8\}$. On the other hand, $\mathcal{C}_2$ is also implemented using four messages that are scheduled as $\{4,0,8\}$, $\{4,1,8\}$, $\{4,2,8\}$, and $\{4,3,8\}$. Using these schedules, we fulfill the resource requirements of both applications, i.e., we allocate four slots in 8 bus cycles for each application. Here, we assign two slot ids, i.e., 3 and 4. Note that these two slot ids cannot be used any further by a message that requires a repetition rate of 1, 2 or 4, despite not being fully utilized. Messages with higher repetition rates, i.e., 8, 16, $\cdots$, N_{com}, can only be accommodated.

- *Case 2*: The message schedules for $\mathcal{C}_1$ remain the same as in *Case 1*, while the messages for $\mathcal{C}_2$ are sent using the following schedules: $\{3,4,8\}$, $\{3,5,8\}$, $\{3,6,8\}$, and $\{3,7,8\}$. Again, we allocate 4 slots in 8 bus cycles for each application to fulfill the resource requirements. Here, however, we assign only one slot id, i.e., 3. Although slot id 3 cannot be assigned to any further messages as it is fully utilized, slot id 4 is free to be allocated to any message irrespective of its repetition rate.

Thus, in *Case 2*, FlexRay static segment is more extensible as it can accommodate messages with different requirements [242, 243] in contrast to *Case 1*, where messages with only certain periodicity can be accommodated. Note that the slot requirements of a bimodal controller in the static segment are not exactly periodic. For example, both $\mathcal{C}_1$ and $\mathcal{C}_2$ require to stay in the high-cost mode for 50% of the control instances, however, the requirement is not periodic in a general sense. If these requirements were periodic and we could have used two messages for $\mathcal{C}_1$ and $\mathcal{C}_2$ that are scheduled as $\{3,0,2\}$ and $\{4,0,2\}$ respectively, then such a schedule would have been more extensible as it would have allowed messages with repetition rates of 2, 4, $\cdots$, N_{com}. Thus, the main issue here is that the aperiodic message transmissions in the static segment as required by the bimodal controllers cannot be easily interleaved with periodic messages using the same slot id. Therefore, we try to derive a static schedule that interleaves the messages for the bimodal controllers and correspondingly uses as less slot ids as possible. The main goal is to maximize the number of free slot ids that can be used for non-control or future messages with different kinds of requirements.

Slot allocation problem: We formulate a Satisfiability Modulo Theories (SMT) problem to derive a static allocation of slots to applications. In this problem, we consider the resource requirements of each application as a constraint while minimizing the total number of slot ids assigned to the applications.

For an application $\mathcal{C}_i$ with resource requirements given by $n_{T,i}$ and $n_{R,i}$, we consider $n_{R,i}$ boolean variables denoted as $\{\gamma_{i,j} \in \{0,1\} | 1 \leq j \leq n_{R,i}\}$. If $\gamma_{i,j} = 1$ then it denotes that a TDMA slot is assigned to the application in its j-th control instance and thereafter every $n_{R,i}$ instances. That is, if $\gamma_{i,j} = 1$ then the controller is in the high-cost mode for the $(j + k \cdot n_{R,i})$-th instance, where k is a non-negative integer, i.e., $k \in \mathbb{Z}^*$. Using these variables, we can formulate constraints for each application to ensure that the resource requirements are met as follows:

$$\sum_{j=1}^{n_{R,i}} \gamma_{i,j} = n_{T,i}, \quad \forall\ \mathcal{C}_i \in \mathbb{C}. \tag{5.12}$$

$$\bigvee_{j=1}^{n_{R,i}} \left(\left(\bigwedge_{k=j}^{\min(j+n_{T,i}-1, n_{R,i})} \gamma_{i,k} \right) \wedge \left(\bigwedge_{k=1}^{\max(0, j+n_{T,i}-1-n_{R,i})} \gamma_{i,k} \right) \right) = 1, \quad \forall\ \mathcal{C}_i \in \mathbb{C}. \tag{5.13}$$

Here, Eq. (5.12) implies that exactly $n_{T,i}$ slots are allocated for $n_{R,i}$ consecutive control instances, while Eq. (5.13) ensures that the $n_{T,i}$ slots that are allocated will be used in consecutive instances. For example, when $n_{T,i} = 2$ and $n_{R,i} = 4$, Eq. (5.12) gives:

$$\gamma_{i,1} + \gamma_{i,2} + \gamma_{i,3} + \gamma_{i,4} = 2,$$

and Eq. (5.13) formulates:

$$(\gamma_{i,1} \wedge \gamma_{i,2}) \vee (\gamma_{i,2} \wedge \gamma_{i,3}) \vee (\gamma_{i,3} \wedge \gamma_{i,4}) \vee (\gamma_{i,4} \wedge \gamma_{i,1}) = 1.$$

That is, the first constraint will ensure that exactly two out of the four boolean variables are equal to 1, and the second constraint will only allow two consecutive variables to take the value of 1. Thus, satisfying these two constraints will guarantee that the resource requirements are met, as obtained from Stage 1.

Towards formulating the optimization objective, let us assume that we can get a solution using a minimum of n_{id} number of slot ids. Here, we consider that all applications in $\mathbb{C}$ have the same sampling period. However, our formulation can easily be extended to a more general case. With all applications having the same sampling periods, we can formulate a constraint to ensure that no more than n_{id} applications must be in the high-cost mode at the same time as follows:

$$\sum_{i=1}^{|\mathbb{C}|} \gamma_{i,\text{mod}(k-1,n_{R,i})+1} \leq n_{id}, \quad \forall\ 1 \leq k \leq n_R^* = \max_{\forall\ \mathcal{C}_i \in \mathbb{C}} n_{R,i}. \tag{5.14}$$

Note that we determine the maximum among all $n_{R,i}$s obtained from Stage 1 denoted as n_R^*. We formulate a constraint for each control instance till n_R^*. This is because $n_{R,i}$ is a power of 2, and therefore, the least common multiple of all $n_{R,i}$s, that determines the periodicity of the schedule, is also given by the maximum among these $n_{R,i}$s. That is, the schedule repeats after the n_R^* control instances. Now, for each instance till the n_R^*-th instance, we formulate a constraint to ensure that the number of control applications that will use a TDMA slot for communication is less than or equal to n_{id}.

Let us consider an example with three applications $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ with $n_{R,1} = 4$, $n_{R,2} = 8$, and $n_{R,3} = 16$. Here, we need to formulate 16 constraints as the maximum among $n_{R,1}$, $n_{R,2}$, and $n_{R,3}$ is equal to 16. Here, the constraint for the 13-th instance can be written as follows:

$$\gamma_{1,1} + \gamma_{2,5} + \gamma_{3,13} \leq n_{id}.$$

For $\mathcal{C}_1$, if a slot is allocated for the first instance given by $\gamma_{1,1}$ then it will also be allocated for the 13-th instance. Similarly, for $\mathcal{C}_2$, slot allocation for the 5-th instance given by $\gamma_{2,5}$ will be repeated for the $13-th$ instance. Satisfying the constraints in Eq. (5.14) will ensure that the maximum number of applications that are scheduled on the static segment in the same bus cycle cannot be greater than n_{id}. Accordingly, our objective is to minimize the value of n_{id}.

Summary: In Stage 2, we formulate an SMT problem considering constraints derived from the resource requirements of the applications obtained from Stage 1, while minimizing the number of slot ids assigned to the applications. The optimization problem can be written as follows:

$$\textbf{Minimize } n_{id} \textbf{ s.t. } \text{Eq. (5.12), Eq. (5.13), and Eq. (5.14).} \tag{5.15}$$

5.4 Dynamically Scheduling the Bimodal Controllers

In the last section, we have derived a static schedule for the bimodal controllers such that the control performance requirements are met. However, note that in the static scheduling scheme, the controller will always use the minimum amount of high-quality resources required to meet the control requirements. In the event when only one application is experiencing disturbance, it is not possible to provision more high-quality resources to the application such that the disturbance is rejected faster even though other applications do not require the resources as they are in the steady state. Thus, there might be a scope to improve the average control performance of applications using the same amount of resources if online re-allocation of resources is allowed.

Towards this, we further propose a dynamic scheduling strategy for the bimodal controllers in this section. Here, we first study the physical dynamics of the controlled plant to derive timing constraints from control specifications based on which we determine a switching scheme. For the case where multiple application are sharing one TDMA slot, we devise a dynamic priority based arbitration strategy for the scheduler. We further show using a motivational example how to calculate the timing parameters for a given requirement on control performance and further demonstrate the effectiveness of our proposed scheduling strategy.

5.4.1 Switching Strategy

As discussed in Section 5.2.1, we consider a problem setting where each control application $\mathcal{C}_i$ can send the control data either on the static segment or on the dynamic segment of a FlexRay communication cycle. Using the bimodal controller studied in Section 5.2.2, $K_{HC,i}$ is used to calculate the control input when the communication is carried out using a TDMA slot in the static segment, while $K_{LC,i}$ is applied when FTDMA communication is used. As discussed in Section 5.3.1, for a settling time requirement given by J_i^* and $J_{HC,i} < J_i^* < J_{LC,i}$, we can save the TDMA slots in the static segment without jeopardizing the control requirements by switching between the high-cost and the low-cost modes. Here, $J_{HC,i}$ (or $J_{LC,i}$) is the settling time when the controller operates exclusively in the high-cost mode (or in the low-cost mode).

In this work, we propose a switching scheme for the bimodal controllers, as shown in Figure 5.6, that enables sharing a TDMA slot among multiple applications. According to the proposed scheme, when the system is in the steady state, it is sufficient to apply a stable control, i.e., the controller should not force the plant out of the stability threshold. Now, if the closed-loop system in the low-cost mode, as given by Eq. (5.6), is stable, then the controller can stay in the low-cost mode when the system is in the steady state. Thus, to realize our proposed switching scheme, it is necessary to design $K_{LC,i}$ such that the eigenvalues of the closed-loop state transition matrix $\Phi_{LC,i}$ for the low-cost mode are within a unit circle in the complex plane. Note that using the low-cost mode for the steady state control will not impact the control performance as it is measured by the settling time of the system after a disturbance.

Now, when a disturbance arrives, the controller needs to reject the disturbance within a certain time threshold given by J_i^*. As $J_{HC,i} < J_i^* < J_{LC,i}$, the controller needs to switch to the high-cost mode for a certain number of control instances to meet the requirements. Thus, on the arrival of a disturbance, the controller tries to switch to the high-cost mode. Switching to the high-cost mode here means that the controller must get a TDMA slot to send its control

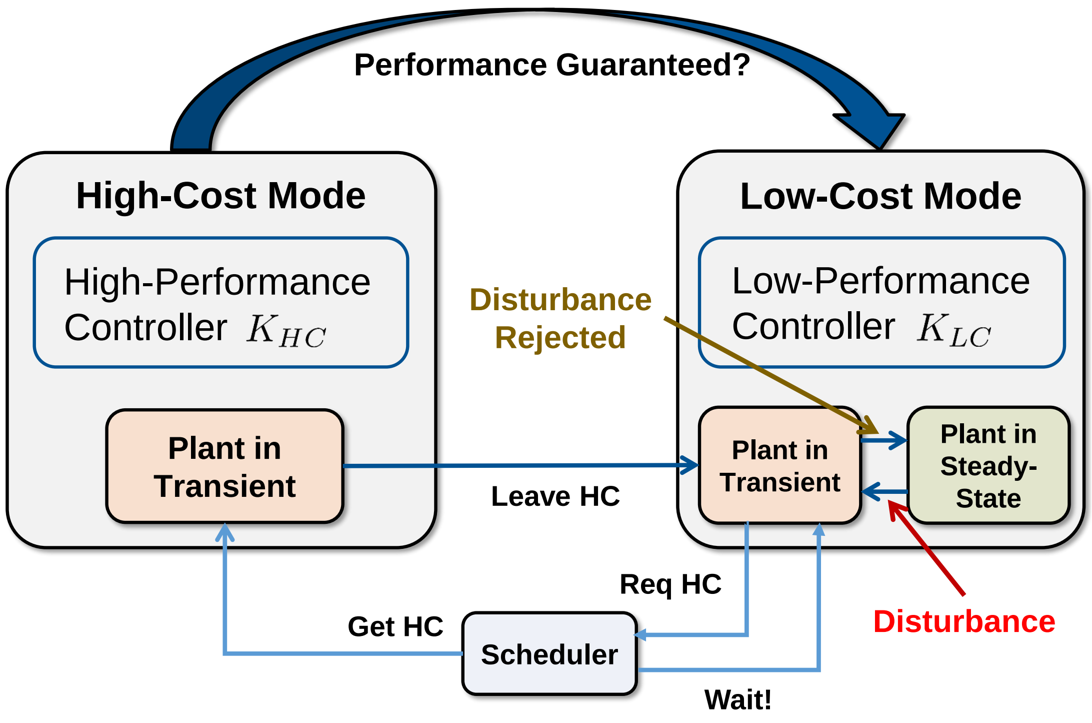


Figure 5.6: The proposed dynamic switching scheme for bimodal controllers. When the controlled plant is in steady state, the controller stays in the low-cost mode and on disturbance, it requests for a switch to the high-cost mode. When the controller switches to the high-cost mode, it stays there until the control performance can be guaranteed.

data on the FlexRay static segment. Now, when multiple applications share a slot, there must be a scheduler that allocates the slot to the application. Therefore, the application experiencing disturbance must request the scheduler for the TDMA slot. Depending on the scheduler decision, the application can either get the slot immediately on request or it might have to wait for a certain number of samples $n_{w,i}$.

Now, for a certain number of samples $n_{w,i}$ that the controller has waited in the low-cost mode after a disturbance, there is a minimum number of samples $n_{T,i}^{-}$ for which the controller must stay in the high-cost mode continuously to meet the settling time requirement. Therefore, once a controller switches to the high-cost mode after waiting for n_w samples, it stays there for a minimum of $n_{T,i}^{-}$ samples. During this time, the scheduler cannot allow other applications to switch to the high-cost mode that share the same slot even if they are disturbed. For a certain $n_{w,i}$, there is also a maximum number of samples $n_{T,i}^{+}$ until which if the controller stays in the high-cost mode it will improve the control performance. However, staying in the high-cost mode beyond $n_{T,i}^{+}$ samples will not improve the control performance further, and thus, it does not make sense to use high-quality resources beyond $n_{T,i}^{+}$ samples. When a controller have stayed in the high-cost mode for $n_{T,i}^{+}$ samples, it automatically switches back to the low-cost mode. On the other hand, note that a controller can also be forced by the scheduler to switch back to the low-cost mode after $n_{T,i}^{-}$ samples in the high-cost mode. This is possible because the application has already fulfilled the requirements and the high-quality TDMA slot might be required by another application experiencing disturbance. It is to be also noted that once a controller switches back to the low-cost mode, it stays there at least until the arrival of the next disturbance. That is, the controller does not switch back to the high-cost mode even in the case where the high-quality resource (i.e., the TDMA slot) is free, the disturbance is not completely rejected, and the controller has not yet spent n_T^+ samples in the high-cost mode.

Given the requirement J_i^* and the control gains $K_{HC,i}$ and $K_{LC,i}$ for the high-cost and the low-cost modes respectively, we can simulate the closed-loop system for all possible switching sequences allowed by the proposed strategy. Thus, we can precalculate n_T^- and n_T^+ for all possible n_w. Note that we can choose n_w with a certain granularity to enhance scalability. This also allows to implement the control strategy using less memory. There is a trade-off between conservativeness on one hand and scalability and memory requirements on the other.

5.4.2 Scheduling Policy

As mentioned earlier, we consider a case where multiple bimodal controllers share a TDMA slot. This essentially means that only one controller can operate in the high-cost mode at any point in time. Therefore, a scheduler selects the application based on a certain allocation policy that will get the slot when multiple applications are contesting for the slot. In this work, we propose a dynamic priority based policy for slot allocation. According to our proposed scheduling policy, each application is assigned priority dynamically based on the current state of the application. The scheduler allocates the slot to the application that has the highest priority among all the contesting applications.

First, the scheduler needs to identify the applications that are contesting for the slot. The applications that are in the steady state and not experiencing any disturbance are obviously not contesting for the slot. However, it is also not true that all applications that are currently in the transient state, are requesting for the slot. For example, let us consider a case where an application $\mathcal{C}_1$ has switched to the high-cost mode after a disturbance and stayed in that mode for the mandatory $n_{T,1}^-$ samples. Now, the application is forced to switch back to the low-cost mode because another application $\mathcal{C}_2$ is experiencing a disturbance. Now, even if $\mathcal{C}_1$ has not fully rejected the disturbance when $\mathcal{C}_2$ has completed the minimum $n_{T,2}^-$ samples in the high-cost mode, $\mathcal{C}_1$ will not be considered again by the scheduler for a switch back to the high-cost mode. That is, as shown in Figure 5.2 and Figure 5.6, for one disturbance event, the controller switches from the low-cost mode to the high-cost mode and then again back to the low-cost mode where the remaining disturbance (if any) is rejected. Thus, the arrival of a disturbance on an application triggers a switching request from the application to the scheduler. Once an application switches back to the low-cost mode after staying in the high-cost mode for the required number of samples, the request is withdrawn from the scheduler.

Among the applications that are contesting for the slot, each application has a priority. The highest priority is assigned to the application $\mathcal{C}_i$ that is currently using the slot and has not yet stayed in the high-cost mode for $n_{T,i}^-$ samples. Thus, such an application cannot be preempted by any other application waiting for the slot. On the other hand, the lowest priority is assigned to the application that is currently using the slot but has stayed in the high-cost mode for more than $n_{T,i}^-$ samples but less than $n_{T,i}^+$ samples. This application will immediately switch back to

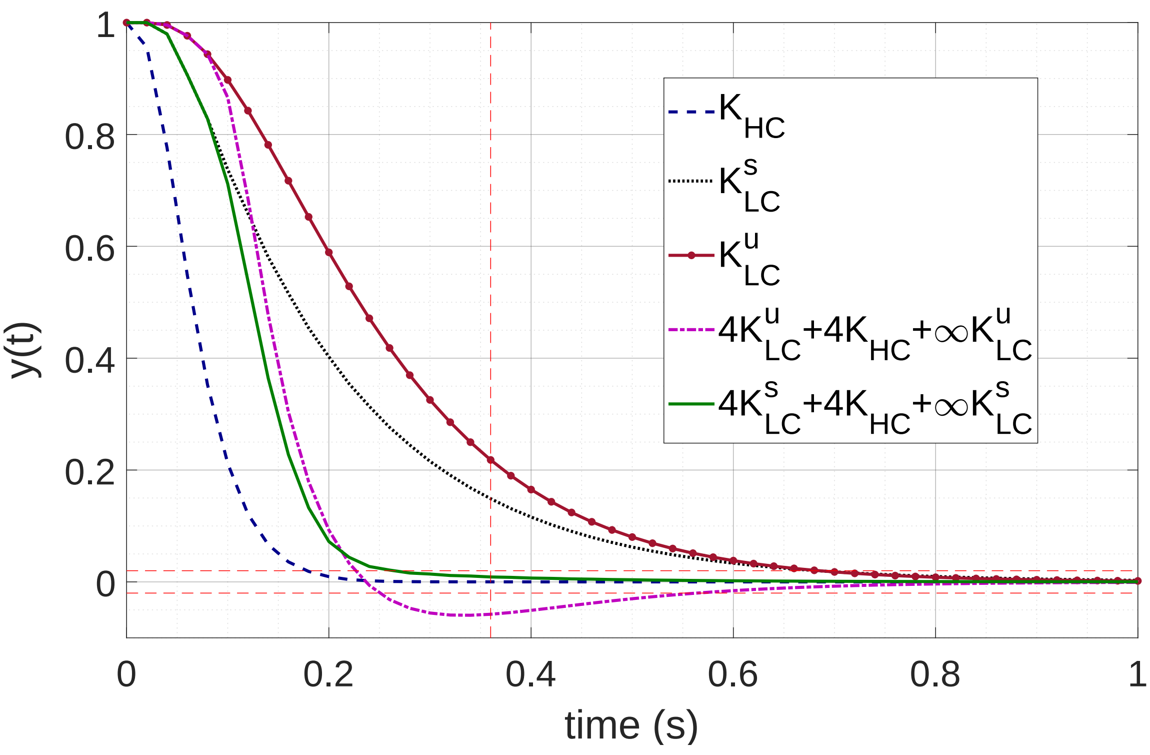


Figure 5.7: Control responses for different switching strategies. The blue dashed line shows the response when the controller only operates in the high-cost mode and uses K_{HC}. The black dotted line and the brown marked-solid line show the responses when the controller operates only in the low-cost mode using the control gains K^s_{LC} and K^u_{LC} respectively. The green solid line and the pink dash-dotted line show the control responses when bimodal controllers, i.e., $\{K_{HC}, K^s_{LC}\}$ and $\{K_{HC}, K^u_{LC}\}$ respectively, are used.

the low-cost mode when a new application has requested to switch to the high-cost mode after being disturbed.

For all other applications waiting for the slot, the priorities are assigned based on their criticality. For each application, there is a maximum number of samples $n^*_{w,i}$ for which it can wait in the low-cost mode after a disturbance without violating the control performance. If an application waits for more than $n^*_{w,i}$ samples in the low-cost mode after a disturbance then it will not satisfy the control performance irrespective of the number of samples it stays in the high-cost mode thereafter. Now, the scheduler uses this value to determine the criticality of the application. That is, it assigns priority to an application depending on the number of samples remaining till $n^*_{w,i}$, i.e., $n^*_{w,i} - n_{w,i}$. Lower the number of samples remaining within which the application must be provided with the TDMA slot, more critical it is, and therefore, higher is its priority. That is, during the arbitration, among all contesting applications, the application that is nearest to having waited the maximum number of permissible samples, will get the slot, and therefore, will switch to the high-cost mode.

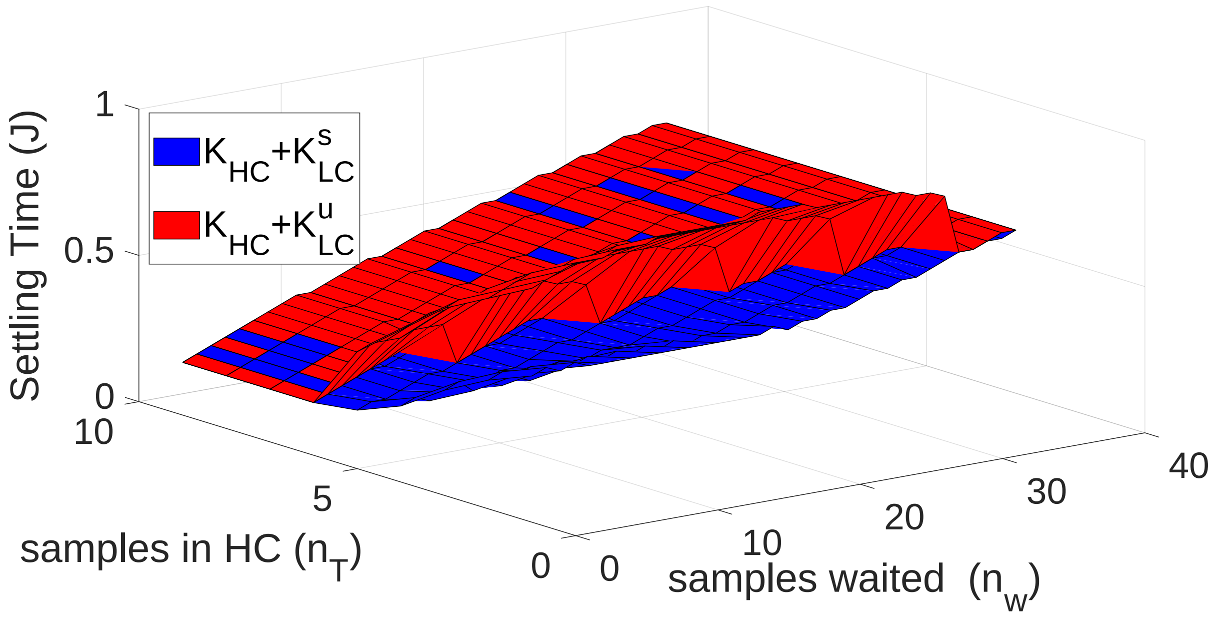


Figure 5.8: Impact of switching stability on control performance. For all possible switching instants defined by n_w and n_T, the settling time is plotted for two bimodal controllers comprising $\{K_{HC}, K^s_{LC}\}$ and $\{K_{HC,K^u_{LC}}\}$ respectively. For the same switching combination, the settling time is alway lower for the former bimodal controller than the latter.

5.4.3 A Motivational Example

Again, we review the example of the DC motor position control system that is described by the continuous-time state-space model given in Eq. (5.8). The control requirements are as outlined in Section 5.3.2. Three controllers K_{HC}, K^s_{LC}, and K^u_{LC} are designed for this system as explained in Section 5.3.2 and given in Eq. (5.9), Eq. (5.10), and Eq. (5.11) respectively.

We plot control responses for different switching strategies in Figure 5.7. Here, the responses for K_{HC}, K^s_{LC}, and K^u_{LC} are exactly same as the ones shown in Figure 5.5. Note that when K_{HC} is used then communication is carried out on the static segment, and for K^s_{LC} and K^u_{LC}, dynamic segment is used for the communication. The control responses using the switched control for two different cases are also shown. In both cases, the controller stays in the low-cost mode for four control instances after a disturbance, followed by four instances in the high-cost mode, before returning to the low-cost mode again. As expected, better settling times are obtained by using the high-cost mode for four samples as compared to none. However, when K^s_{LC} and K_{HC} are used it gives a better settling time of $0.28\,\text{s}$ (green solid line), as compared to $0.58\,\text{s}$ (pink dash-dotted line) that is obtained when K^u_{LC} and K_{HC} are used. Note that both K^s_{LC} and K^u_{LC} give the same performance when applied individually. Thus, the settling time requirement is not met when K^u_{LC} is used in the low-cost mode while it is met with K^s_{LC}. This difference in performance again suggests that, for our proposed scheduling strategies, the two controllers used in the high-cost and the low-cost modes respectively must satisfy the switching stability constraint as per Eq. (5.7).

We have also simulated the system for all possible switching combinations considering both pairs of control gains, i.e., $K_{HC} + K_{LC}^s$ and $K_{HC} + K_{LC}^u$. J as a function of n_w (number of instances the application has waited for the high-cost mode after a distrubance) and n_T (number of instances for which the controller switches to the high-cost most during a disturbance) is plotted in Figure 5.8. The result shows that the system design without considering switching stability is less efficient in terms of the control performance.

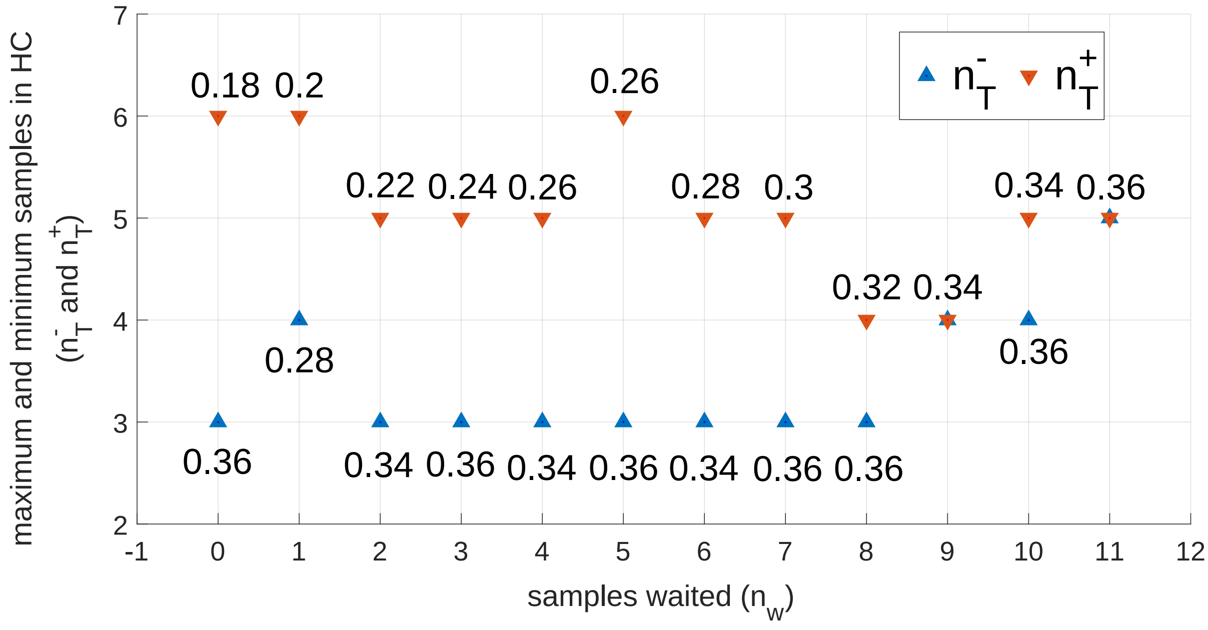


Figure 5.9: Resource requirements for the proposed dynamic scheduling of the TDMA slots. Given a settling time requirement $J^* = 0.36\,\text{s}$, this plot shows the minimum and the maximum number of samples (n_T^- and n_T^+) that the controller can stay in the high-cost mode for a certain number of samples it has waited in the low-cost mode after a disturbance. Each point is annotated with the corresponding settling time value in seconds.

Using K_{HC} and K_{LC}^s, the system is simulated for all possible switching combinations considering $J^* = 0.36\text{s}$, and the results are shown in Figure 5.9. In the figure, Note that the best settling time for a certain n_w is achieved when the controller stays in the high-cost mode for n_T^+ instances and it is non-decreasing with increase in n_w. For $n_w = 0$, switching from high-cost mode to the low-cost mode after six samples can still result in the same performance as for the case when the controller only operates in the high-cost mode. Thus, it is pessimistic to stay in the high-cost mode till the whole disturbance is rejected. It can be observed that n_T^- and n_T^+ varies with n_w. All points between n_T^- and n_T^+ also satisfy the settling time requirement.

5.5 Resource Dimensioning for Dynamic Scheduling

In the last section, we have proposed a strategy that dynamically allocates a TDMA slot to applications based on requirements. Now, an important research question is: *Given a pool of applications and the proposed strategy what is the minimum number of slots required to* *guarantee control performance of all applications?* Towards this question, we propose a nested two layer technique. In the outer layer, we use a first-fit heuristic to map applications to slots, while in the inner layer we verify the control performance for a set of applications mapped on to one slot. Note that here we obtain a many-to-one mapping of applications to slots. That is, one application is mapped on to one specific slot that it can use when required, and more than one application can be mapped to one slot. Here, we do not use a many-to-many mapping of applications to slots, where an application can use any slot if available. This is because such a mapping might not be as resource-efficient as the many-to-one mapping.

Let us consider four applications, out of which two applications $\mathcal{C}_1$ and $\mathcal{C}_2$ have $n^*_{w,i} = 8$ and $n^-_{T,i} = 3$ and the other two applications $\mathcal{C}_3$ and $\mathcal{C}_4$ have $n^*_{w,i} = 20$ and $n^-_{T,i} = 10$. Here, we assume a constant $n^-_{T,i}$ for all applications. Now, using many-to-one mapping, we can map $\mathcal{C}_1$ and $\mathcal{C}_2$ to one slot id and $\mathcal{C}_3$ and $\mathcal{C}_4$ to another slot, and such a mapping will guarantee that no applications will have to wait more than their given $n^*_{w,i}$ number of samples. However, a many-to-many mapping of these applications to two slots will violate the requirements. Consider a case where $\mathcal{C}_3$ and $\mathcal{C}_4$ are disturbed together followed by $\mathcal{C}_1$ and $\mathcal{C}_2$ after one sample. Thus, the two slots will be first occupied by $\mathcal{C}_3$ and $\mathcal{C}_4$ and they will use the slots for 10 samples. However, by that time $\mathcal{C}_1$ and $\mathcal{C}_2$ will have already waited for 9 samples that violates their requirements. Thus, with many-to-many mapping these applications would require three slots to fulfill their performance requirements, which is 50% more resources than in case of many-to-one mapping.

In this section, we will consider many-to-one mapping of applications to slots and explain the proposed nested two layer technique for accurate dimensioning of TDMA slots.

5.5.1 Inner Layer: Verifying Control Performance

Given the switching bimodal control strategy and the scheduling policy, in this layer, we verify that all applications mapped on a TDMA slot in the static segment will meet their requirements in all possible scenarios. The problem can be reformulated to verify that each application $\mathcal{C}_i$ switches to the high-cost mode before $n^*_{w,i}$ has elapsed. Note that this problem is similar to verifying schedulability of multiple tasks running on a processor. However, an important difference is that the minimum time an application must stay in the high-cost mode to meet its requirement depends on the time it has waited in the low-cost mode after the disturbance. Thus, this verification problem does not fit into any standard schedulability analysis framework and we use model checking to solve it. Towards this, we propose to model the whole system as a network of timed automata and verify the model using UPPAAL [184]. Here, we abstract the control dynamics using timing variables like $n_{w,i}$, $n^*_{w,i}$, $n^-_{T,i}$ and $n^+_{T,i}$. For each application, $n^*_{w,i}$ and the variation of $n^-_{T,i}$ and $n^*_{T,i}$ with $n_{w,i}$ can be predetermined. Here, the assumption is that the control models are fully known with no uncertainties.

A timed automaton (TA) is a finite state automaton with a finite set of real-valued clocks which progress synchronously, as studied in Section 2.4.4. Different TAs can communicate via shared variables and synchronization channels. Our system model consists of TAs representing the applications, the scheduler and the arbitration policy.

There are two *main challenges* in modeling the system as a network of TAs. (i) The system under study is discrete-time, i.e., disturbances can be sensed only at periodic instants. However, timed automata has continuous-time semantics. Thus, it is challenging to model that the sched-

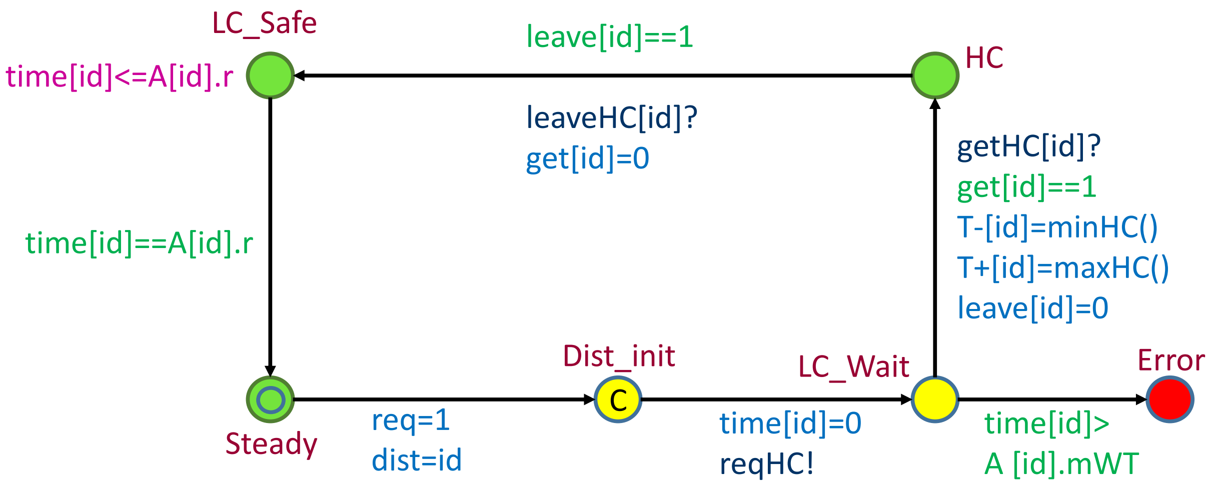


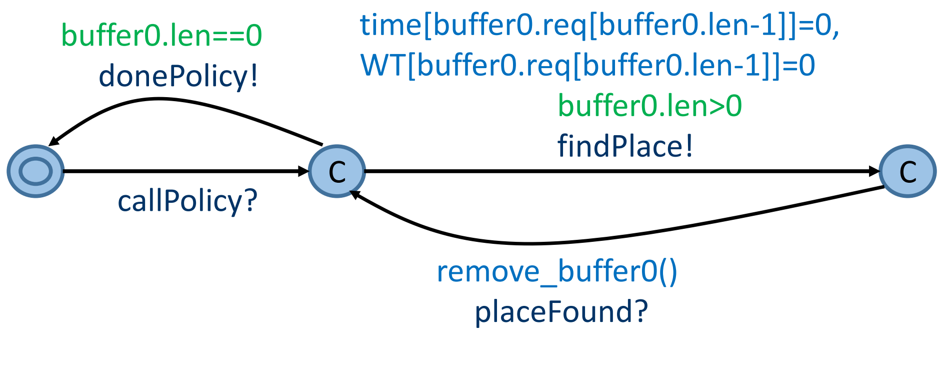
Figure 5.10: An Application automaton.

uler sees multiple slot requests at the same time. (ii) For a certain $n_{w,i}$, the values of $n^-_{T,i}$ and $n^+_{T,i}$ need to be looked up from a precomputed table. Here, n_w can be measured using a clock. However, this clock cannot be used to reference table elements.

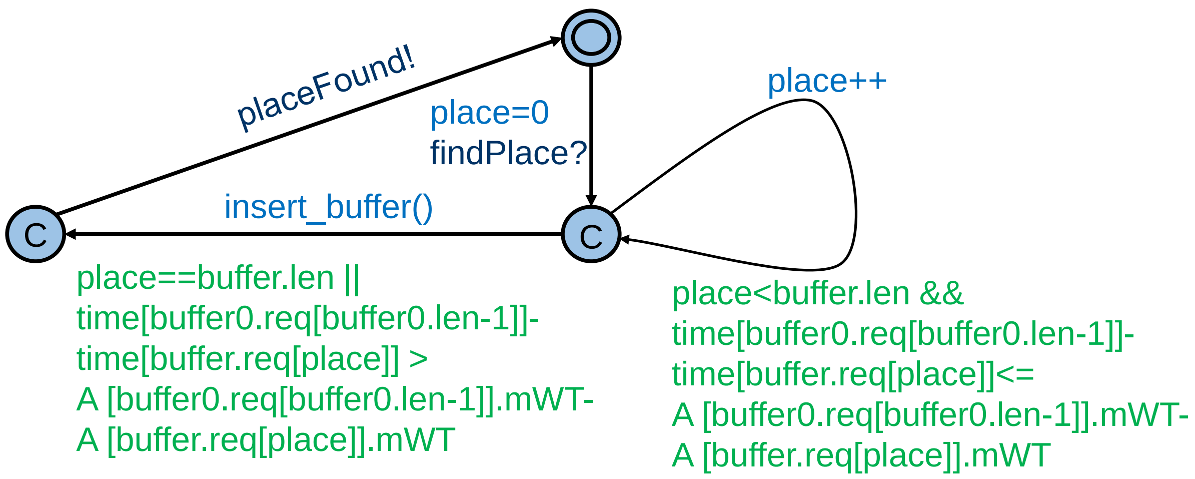
We describe the TAs used to model the system and explain how these challenges are addressed as follows:

- **Application automaton:** Each application automaton has an *id* and uses a clock *time[id]*. As depicted in Figure 5.10, it starts in the *Steady* state. In the event of a disturbance, it requests the scheduler for the TDMA slot using the synchronization channel *reqHC* and moves to the state *LC_Wait*. In this state, it waits for the slot such that it can switch to the high-cost mode. Here, *time[id]* measures the time that has elapsed since the disturbance is sensed. The transition from the *LC_Wait* state to the *Error* state can be taken if the maximum waiting time has elapsed, i.e., *time[id]* is greater than $n^*_{w,i}$. Thus, an application meets the requirement in all scenarios only if it never reaches the *Error* state. On the other hand, slot allocation is notified via the synchronization channel *getHC[id]*. Correspondingly, the automaton takes the transition to the state *HC*. During this transition, the minimum and the maximum number of samples (*T-[id]* and *T+[id]*, i.e., $n^-_{T,i}$ and $n^+_{T,i}$) that the application will stay in the *HC* state is looked up based on the shared variable *WT[id]* that stores $n_{w,i}$. The scheduler preempts the application from the slot via the synchronization channel *leaveHC[id]*. The automaton correspondingly moves to the state *LC_Safe*. It waits here until the minimum disturbance inter-arrival time has elapsed, after which it moves to the *Steady* state again.

- **Automata representing the scheduling policy:** Two nested TAs (*Policy* and *Sort*), as shown in Figure 5.11, implement the scheduling policy. They basically sort the requests from the applications for the use of the TDMA slot and keep them in the order in which it will be served. The scheduler maintains two queues, i.e., *buffer0* and *buffer*. Any requests coming in between two time samples are first stored in *buffer0*. At each time sample, the scheduler invokes these automata to transfer requests from *buffer0* to *buffer* and sort them according to their respectively deadlines. Time does not pass during this operation and requests are



(a)


placeFound!
place=0
findPlace?
place++
insert_buffer()
C
C
place==buffer.len ||
time[buffer0.req[buffer0.len-1]]-
time[buffer.req[place]] >
A [buffer0.req[buffer0.len-1]].mWT-
A [buffer.req[place]].mWT
place<buffer.len &&
time[buffer0.req[buffer0.len-1]]-
time[buffer.req[place]]<=
A [buffer0.req[buffer0.len-1]].mWT-
A [buffer.req[place]].mWT

(b)

Figure 5.11: Automata representing the scheduling policy. (a) The Policy automaton. (b) The Sort automaton.

served from *buffer* only after this operation. This partially addresses the first challenge where scheduler sees the disturbances that arrive during a period together at the next time instant. Here, *Policy* automaton checks for new requests in *buffer0*. For each new request in *buffer0*, *Sort* is invoked to insert the request correctly in *buffer*. Note that when a request is transferred from *buffer0* to *buffer*, the corresponding clock *time[id]* and the waiting time counter *WT[id]* are reset. This marks the time sample when the scheduler sees the disturbance for the first time. To place a new request correctly, *Sort* iterates through the requests in *buffer* one by one and compares their relative deadlines to that of the incoming request.

- **Scheduler automaton:** It implements the scheduler and is shown in Figure 5.12. It can register asynchronous requests from applications in between time samples via the synchronization channel *reqHC*. These requests are stored with the application *id* in *buffer0*. Besides, it also invokes a sequence of operations at every time sample based on a clock *x* which resets every time unit. This is done to implement a discrete-time scheduler and thereby addresses the first

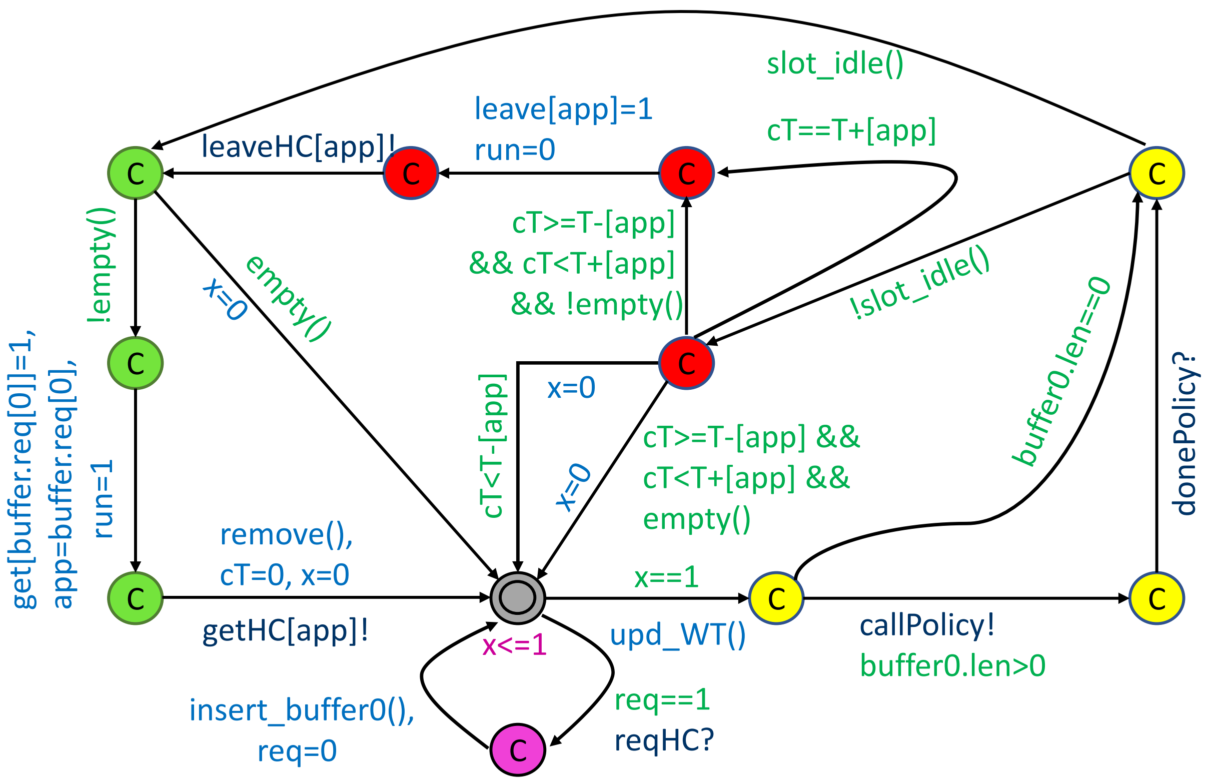


Figure 5.12: The Scheduler automaton.

challenge. At each sample, the scheduler first increments the waiting time counter *WT[id]* of each application in *LC_Wait* state. *WT[id]* is used to reference the look-up table (that stores $n^-_{T,i}$ and $n^+_{T,i}$ as a function of $n_{w,i}$) during the transition of an *Application* automaton from *LC_Wait* to the *HC* state. This addresses the second challenge. A more intuitive way of addressing the second challenge is by adding a self transition every time unit at *LC_Wait* state in the *Application* automaton. However, it turns out to be very inefficient due to an additional clock in each *Application* automaton and also numerous additional state transitions. Now, after updating the wait time counters, the scheduler automaton calls *Policy* and *Sort* if *buffer0* is not empty. After the requests are arranged in *buffer*, it checks if the slot is idle. When the slot is free and the *buffer* is not empty, the first application in the *buffer* gets the slot and the corresponding request is removed from the *buffer*. The application is simultaneously notified via the synchronization channel *getHC[·]* and the clock *cT* is reset. On the other hand, if the slot is occupied then the scheduler checks if the allocated application can be forced to switch back to the low-cost mode. Here, it uses the clock *cT* and the shared variable T-[·] to check if the application that is currently using the slot has stayed in the high-cost mode for the minimum number of required samples. If yes, the application is forced to switch from the high-cost mode via the synchronization channel *leaveHC[·]*. Thereafter, the free slot can be assigned to a waiting application (if any). After completion of all the operations, the clock *x* is reset.

Algorithm 6: Mapping algorithm to slots using a first-fit heuristic.

```
Input    : $\{t_{a,i}, n^*_{w,i}, \mathrm{f}_{T,i}(n_{w,i}) | \forall\ \mathcal{C}_i \in \mathbb{C}\}$
Output   : $Map$
1  $\mathbb{C}_{srt}$ = SortAscending$\Big(\{n^*_{w,i}, \mathrm{f}_{T,i}(n_{w,i}) | \forall\ \mathcal{C}_i \in \mathbb{C}\}\Big)$;
2  $Map = [\{\mathbb{C}_{srt}[1]\}]$;
3  for $j \leftarrow 2$ to $|\mathbb{C}_{srt}|$ do
4      $\mathcal{C}_j = \mathbb{C}_{srt}[j]$;
5      for $k \leftarrow 1$ to $|Map|$ do
6          $flag$ = Verify$\Big(\{t_{a,i}, n^*_{w,i}, \mathrm{f}_{T,i}(n_{w,i}) | \forall\ \mathcal{C}_i \in Map[k] \cup \mathcal{C}_j\}\Big)$;
7          if $flag == 1$ then
8              $Map[k] = \mathcal{C}_j \cup Map[k]$;
9              break;
10         else if $k == |\mathbb{C}_{srt}|$ then
11             $Map = [Map, \{\mathcal{C}_j\}]$;
12             break;
13         end
14     end
15 end
16 return $Map$;
```

Verification: The whole system is schedulable or the performance requirements of all applications will be met if none of the applications ever reach its *Error* state. Thus, the problem boils down to reachability analysis. This can be formulated in UPPAAL as follows:

$$\text{A[] not (C1.Error || C2.Error ||} \cdots \text{|| Cn.Error)} \tag{5.16}$$

5.5.2 Outer Layer: Mapping Applications to Slots

In the proposed nested two layer approach for the accurate dimesnioning of TDMA slots, we use a first-fit heuristic in the outer layer to map applications to slot ids. The first-fit heuristic that we use is outlined in Algorithm 6. The algorithm takes as input the timing information for each application $\mathcal{C}_i$, i.e., the interarrival time for disturbances $t_{a,i}$, the maximum number of samples $n^*_{w,i}$ that $\mathcal{C}_i$ can wait in the low-cost mode after a disturbance, and a function that determines the value of the minimum and the maximum number of samples, $n^-_{T,i}$ and $n^+_{T,i}$, that $\mathcal{C}_i$ can stay in the high-cost mode after waiting for $n_{w,i}$ samples. The output of the algorithm is the minimum number of slots ids required to meet the requirements of all applications as well as the groups of applications that must use the same slot id. The output variable Map is an array of sets of application, where each set is a group of applications that will use the same slot id based on the proposed scheduling policy and switching control strategy.

In line 1, the algorithm sorts the applications in the ascending order with respect to $n^*_{w,i}$ and this ordered list is denoted as $\mathbb{C}_{srt}$. The less number of samples the application can wait in the low-cost mode, lower is it position in the sorted list. Now, for two applications $\mathcal{C}_i$ and $\mathcal{C}_j$ with $n^*_{w,i} = n^*_{w,j}$, the order is decided based on the maximum value of $n^-_{T,i}$ and $n^-_{T,j}$ denoted as $n^*_{T,i}$ and $n^*_{T,j}$ respectively. That is, if $n^*_{T,i}$ is lower than $n^*_{T,j}$ then $\mathcal{C}_i$ comes before $\mathcal{C}_j$ is the list. In line 2, the algorithm maps the first application in the ordered list to the first slot id, i.e., $\mathbb{C}_{srt}[1]$ is added to the first set in Map.

Using the for loop in lines 3-15, we traverse through the remaining applications in $\mathbb{C}_{srt}$ in order and map each of them to a slot id. Let us take the application $\mathcal{C}_j$ in the j-th position in $\mathbb{C}_{srt}$ (line 4). Now, we traverse the sets of applications in Map one by one (lines 5-14). In the k-th iteration, we check if $\mathcal{C}_j$ and the applications in the k-th set of Map can satisfy the requirements using only one slot id (line 6). Here, we build a network of TAs comprising *Application* automata for $\mathcal{C}_j$ and the applications in the k-th set of Map, together with the *Scheduler*, *Policy*, and *Sort* automata and we verify the model as per Eq. (5.16). If the control performances can be guaranteed, we add $\mathcal{C}_j$ to the k-th set in Map (line 7-9). Now, if $\mathcal{C}_j$ cannot be feasibly added to any of the existing sets then a new set is created with $\mathcal{C}_j$ as the only application and added to Map (lines 10-13). This essentially means that none of the existing slot ids have enough bandwidth to accommodate $\mathcal{C}_j$ and a new slot id is added where $\mathcal{C}_j$ is mapped. In line 16, the algorithm returns Map.

According to the algorithm, the minimum number of slot ids required by all applications is given by the number of sets in Map. Each set in Map will require a unique slot id.

5.6 Experimental Results

Case Study: We consider a case study comprising 6 control applications. $\mathcal{C}_1$ [241] and $\mathcal{C}_2$ [202] are DC motor position control. $\mathcal{C}_3$ [81], $\mathcal{C}_4$ [202] and $\mathcal{C}_5$ [161] represent DC motor speed control. $\mathcal{C}_6$ [202] is a cruise control. The plant models are provided in Table 5.2. We consider a sampling period $h = 0.02\,\text{s}$. For each application $\mathcal{C}_i$, control gains, $K_{HC,i}$ and $K_{LC,i}$, are synthesized for the high-cost and the low-cost modes respectively (see Table 5.2). Here, $K_{HC,i}$ and $K_{LC,i}$ satisfy the switching stability condition. For each application $\mathcal{C}_i$, the minimum interarrival time $t_{a,i}$ for disturbances and the settling time requirement J^*_i are provided in Table 5.2. Note that these times are given in terms of the number of samples. For example, in case of $\mathcal{C}_1$, interarrival time for disturbances $t_{a,1}$ is equal to 25 samples or $0.5\,\text{s}$, and the settling time requirement is 18 samples or $0.36\,\text{s}$.

Application of bimodal controllers: For the given control gains, $K_{HC,i}$ and $K_{LC,i}$, and the plant model, we can determine the settling times, $J_{HC,i}$ and $J_{LC,i}$, that are obtained when $K_{HC,i}$ and $K_{LC,i}$ are applied individually to the plant and the static and the dynamic segments of FlexRay are used respectively for the communication. These obtained values are provided in Table 5.3. Note that for each application $J_{HC,i} < J^*_i < J_{LC,i}$. Thus, to meet the requirements without over-provisioning the high-quality resources, we can use bimodal controllers and switch between the high-cost and the low-cost modes as defined in Section 5.2.2. This will enable sharing of high-quality resources among applications.

Proposed static scheduling strategy: Towards static scheduling of TDMA slots in the static segment of FlexRay, we first derive the resource requirements for each application using Algorithm 5. The requirement is expressed as the number of consecutive control instances $n_{T,i}$ in

Table 5.2: Specification data for the case study (time is measured in terms of the number of samples)

$\mathcal{C}_i$	Plant Model	$K_{HC,i}$	$K_{LC,i}$	$t_{a,i}$	J_i^*
$\mathcal{C}_1$	Eq. (5.8)	Eq. (5.9)	Eq. (5.10)	25	18
$\mathcal{C}_2$	$A=\begin{bmatrix}0&1&0\\0&-1.0865&8.4872\cdot10^3\\0&-9.9636\cdot10^3&-1.4545\cdot10^6\end{bmatrix}$, $B=\begin{bmatrix}0&0&3.6364\cdot10^5\end{bmatrix}^T$, $C=\begin{bmatrix}1&0&0\end{bmatrix}$	$\begin{bmatrix}0.1198\\-0.0130\\-2.9588\end{bmatrix}^T$	$\begin{bmatrix}0.0864\\-0.0128\\-1.6833\\0.4059\end{bmatrix}^T$	100	25
$\mathcal{C}_3$	$A=\begin{bmatrix}-0.2&0.67\\-10&-100\end{bmatrix}, B=\begin{bmatrix}0\\37000\end{bmatrix}, C=\begin{bmatrix}1&0\end{bmatrix}$	$\begin{bmatrix}0.0500\\-0.0002\end{bmatrix}^T$	$\begin{bmatrix}0.0336\\0.0004\\0.4453\end{bmatrix}^T$	50	20
$\mathcal{C}_4$	Eq.(3.28)	$\begin{bmatrix}100.0000\\15.6226\end{bmatrix}^T$	$\begin{bmatrix}-77.8275\\24.3161\\1.0265\end{bmatrix}^T$	40	19
$\mathcal{C}_5$	$A=\begin{bmatrix}-10&1\\-0.2&15\end{bmatrix}, B=\begin{bmatrix}0\\20\end{bmatrix}, C=\begin{bmatrix}1&0\end{bmatrix}$	$\begin{bmatrix}10.0000\\1.0524\end{bmatrix}^T$	$\begin{bmatrix}-2.4223\\0.7014\\0.2950\end{bmatrix}^T$	25	18
$\mathcal{C}_6$	Eq.(3.31)	15000	$\begin{bmatrix}8125.6\\0.8659\end{bmatrix}^T$	100	20

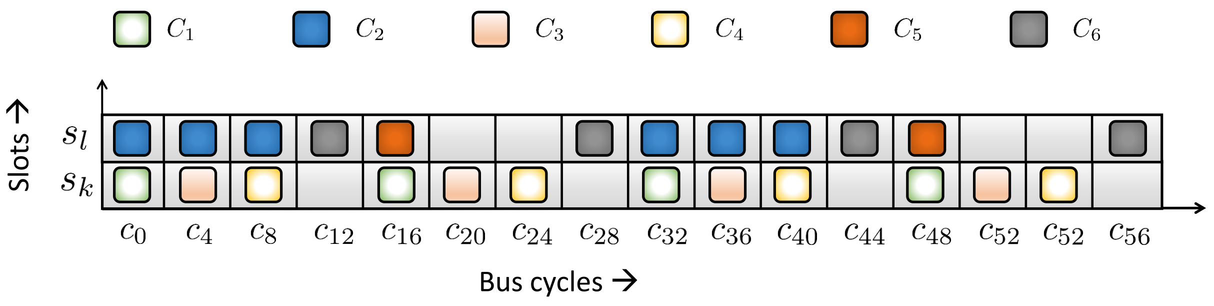

C_1
C_2
C_3
C_4
C_5
C_6
Slots
s_l
s_k
c_0 c_4 c_8 c_{12} c_{16} c_{20} c_{24} c_{28} c_{32} c_{36} c_{40} c_{44} c_{48} c_{52} c_{52} c_{56}
Bus cycles

Figure 5.13: Static allocation of TDMA slots to applications. For each application, the resource requirements, as given by $\{n_{T,i}, n_{R_i}\}$ in Table 5.2, are met.

which the controller must stay in the high-cost mode in a sequence of $n_{R,i}$ instances. For the bus cycle time $T_{bus} = 0.005\,\mathrm{s}$ and the number of configurable communication cycles N_{com}, the values of $n_{T,i}$ and $n_{R,i}$ are determined that are provided in Table 5.3. Based on the values of $n_{T,i}$s and $n_{R,i}$s, we can determine a lower bound on the number of slots ids required as follows:

$$\left\lceil \frac{1}{4}+\frac{3}{8}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{4} \right\rceil = 2.$$

Now, we can apply the optimization given in Eq. (5.15) to obtain a static schedule as shown in Figure 5.13. It is shown in the figure that we can can schedule all application using 2 slot ids,

Table 5.3: Resource requirements of the bimodal controllers for static and dynamic scheduling
(Time is measured in terms of the number of samples)

$\mathcal{C}_i$	$J_{HC,i}$	$J_{LC,i}$	$\{n_{T,i}, n_{R,i}\}$	$n^*_{w,i}$	$n^-_{T,i}$	n^+_T
$\mathcal{C}_1$	9	35	$\{1,4\}$	11	[3,4,3,3,3,3, 3,3,3,4,4,5]	[6,6,5,5,5,6, 5,5,4,4,5,5]
$\mathcal{C}_2$	15	50	$\{3,8\}$	13	[7,7,6,7,6,7,6, 7,6,7,6,7,7,8]	[10,10,9,10,8,9, 9,10,8,8,9,8,8,8]
$\mathcal{C}_3$	10	31	$\{1,4\}$	15	[4,4,4,4,4,4,4,4, 4,4,4,4,4,4,4,4]	[8,8,7,7,7,6,6,6, 6,5,5,5,5,4,4,4]
$\mathcal{C}_4$	10	31	$\{1,4\}$	12	[5,5,5,5,5,5,5, 5,5,5,5,5,5]	[9,8,8,8,8,7,7, 7,7,6,6,6,5]
$\mathcal{C}_5$	10	25	$\{1,8\}$	12	[4,3,3,3,3,3,3, 4,4,4,4,4,4]	[9,8,7,8,7,6,7, 6,5,5,4,4,4]
$\mathcal{C}_6$	11	41	$\{1,4\}$	12	[7,8,7,8,7,8,7, 8,7,8,7,8,8]	[11,11,10,10,10, 10,9,9,9,8,8,8,8]

s_k and s_l. Here, $\mathcal{C}_1$, $\mathcal{C}_3$, and $\mathcal{C}_4$ share the slot id s_k and $\mathcal{C}_2$, $\mathcal{C}_5$, and $\mathcal{C}_6$ share s_l. Note that the bus cycles in Figure 5.13 are numbered as $c_0, c_4, c_8, \cdots$. This is because the sampling period of the applications are $0.02\,\text{s}$ that is four times the bus cycle time of $0.005\,\text{s}$, and therefore, the control instances for these applications come every 4 cycles. The frame schedules for the applications are provided in Table 5.4.

Table 5.4: FlexRay frame schedule assignments for the control applications

$\mathcal{C}_i$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$
$\{s_{c,i}, b_{c,i}, r_{c,i}\}$	$\{s_k, 0, 16\}$	$\{s_l, 0, 32\}$ $\{s_l, 4, 32\}$ $\{s_l, 8, 32\}$	$\{s_k, 4, 16\}$	$\{s_k, 12, 32\}$	$\{s_l, 16, 32\}$	$\{s_l, 12, 16\}$

An important thing to note in the obtained schedule is that the slot ids, s_k and s_l, are not fully utilized. Here, s_k can still be used to send messages with repetition rates of 16, 32, and 64, while s_l only offers the repetition rates of 32 and 64. Thus, with static scheduling, we can use expensive resources only as much is required. Note that if $\mathcal{C}_5$ was scheduled using $\{s_k, 12, 32\}$ instead of $\{s_l, 16, 32\}$, then both s_k and s_l can only offer schedules with repetition rates of 32 and 64. Thus, the schedule will become less extensible. We can also optimize for extensibility in the order of fraction of a slot id, i.e., we can maximize the the number of repetition rates allowed for other real-time or even future messages. Here, after determining the minimum number of slot ids required to fulfill the requirements of all applications, we can try to add a dummy message with the lowest possible repetition rate without increasing the slot id.

In the case study, for example, we have obtained a minimum requirement of 2 slot ids. Now, for all applications, we get that $\frac{1}{4}+\frac{3}{8}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{4}=1.5$ slot ids are required. Thus, in the best-case, a message with $\{n_{T,i}, n_{R,i}\}=\{1,2\}$ might be possible. We try to see if such a dummy message can be scheduled together with the applications in the case study. Here, we do not consider the optimization objective, i.e., minimization of n_{id} as in Eq. (5.15), but rather consider $n_{id}=2$ as given. In this case study, we cannot add such a dummy message without increasing the number of slot ids. Now, we try to add a message with $\{n_{T,i}, n_{R,i}\}=\{1,4\}$ and this is possible using the schedule as shown in Figure 5.13. Thus, in the case study considered here, this is the most extensible schedule as new messages can be added with a repetition rate of 16, 32, and 64. Note that as $h=4\cdot T_{bus}$, the repetition rate $r_{c,i}$ is four times the value of $n_{R,i}$.

Proposed dynamic scheduling strategy: For the given control gains, $K_{HC,i}$ and $K_{LC,i}$, and the plant model, we can simulate the closed-loop system to determine the maximum number of samples $n^*_{w,i}$ that the application $\mathcal{C}_i$ can stay in the low-cost mode after a disturbance without violating the settling time requirement J^*_i. Now for a certain number of samples $n_{w,i}\leq n^*_{w,i}$, we can determine the minimum and the maximum number of sample the application can stay in the high-cost mode, $n^-_{T,i}$ and $n^+_{T,i}$. For each application, $n^*_{w,i}$ and $n^-_{T,i}$s and $n^+_{T,i}$s are provided in Table 5.3. Note that $n^-_{T,i}$s and $n^+_{T,i}$s are stored in an array and array indices give $n_{w,i}$ where $0\leq n_{w,i}\leq n^*_{w,i}$. These arrays can be stored in a memory-efficient way exploiting the fact that $n^-_{T,i}$ and $n^+_{T,i}$ take only a few values. Now, using these values, we apply the nested two-layer technique, as described in Section 5.5 to determine the minimum number of slot ids required to meet the performance of all applications. Using Algorithm 6 as the outer layer and the TA-based verification (explained in Section 5.5.1) in the inner layer, we obtain a mapping of applications to slots as follows: (i) $\mathcal{C}_1$, $\mathcal{C}_5$, $\mathcal{C}_4$, and $\mathcal{C}_3$ are mapped to slot s_k and (ii) $\mathcal{C}_6$ and $\mathcal{C}_2$ share slot s_l.

Comparison with state-of-the-art techniques: In the literature, dynamic scheduling strategies are only studied so far. We apply the two scheduling strategies proposed in [146] on the case study. Note that a recent paper [244] also uses one of the strategies in [146]. According to the strategies in [146], applications get the TDMA slot according to their fixed priorities and then use the slot till the whole disturbance is rejected. The priorities are assigned according to the deadline monotonic scheme where J^*_i is considered as the deadline. The first strategy is similar to the standard non-preemptive deadline monotonic scheme. In the second strategy, the slot requests from the lower priority applications are delayed to reduce the blocking time for higher priority applications. Using the schedulability analysis and the first-fit heuristic proposed in [146], these applications require minimum 4 slots to meet their requirements. The slot partitions are $\{\mathcal{C}_1,\mathcal{C}_5\}$, $\{\mathcal{C}_4,\mathcal{C}_3\}$, $\{\mathcal{C}_6\}$ and $\{\mathcal{C}_2\}$. Thus, our proposed strategies allow a tighter and accurate dimensioning of high-quality TDMA slots and *save* 50% *of slots* compared to the existing state-of-the-art techniques.

Simulation results: For the static scheduling strategy, we consider the case when all applications are disturbed at the same time in the beginning of the FlexRay cycle $\mathcal{C}_0$. The control response for the applications are plotted in Figure 5.14. We can verify from the plot that all applications meet their respective settling time requirements as provided in Table 5.2. Note that for the proposed static scheduling strategy, the performance of a bimodal controller is not

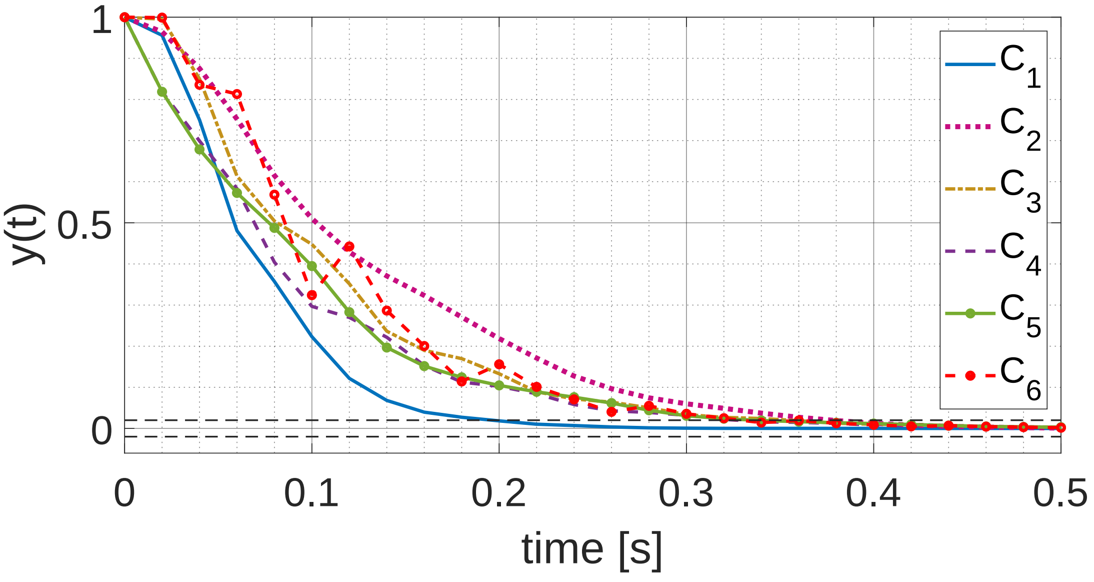


Figure 5.14: Control response of the applications for the proposed static scheduling strategy. Each application meets its settling time requirement as can be verified from this plot.

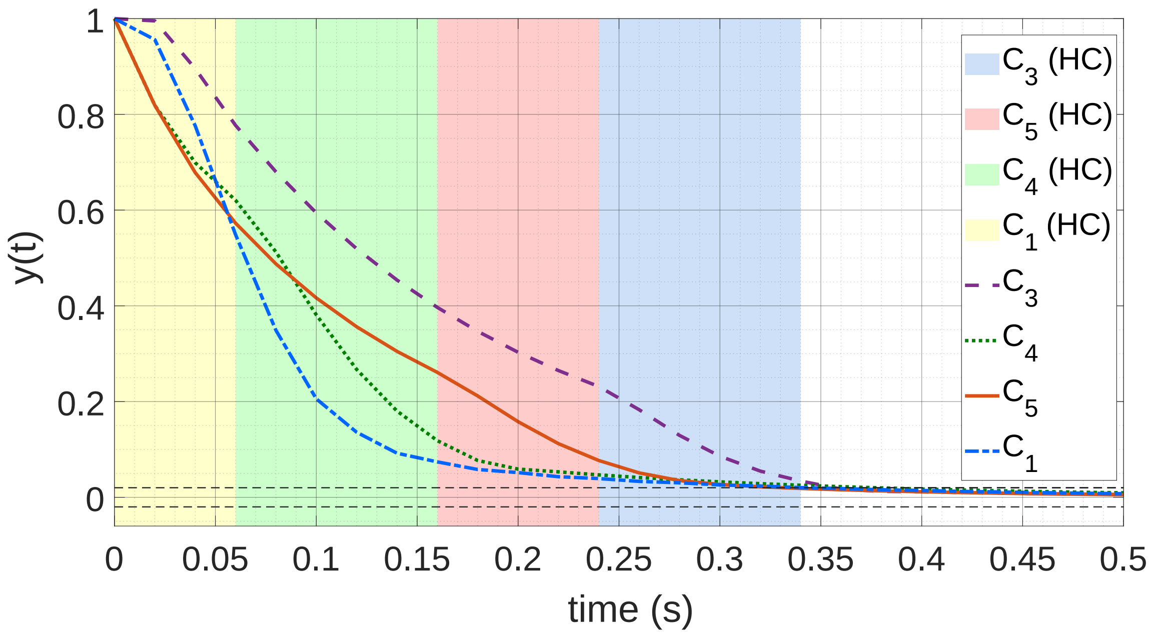


Figure 5.15: Control responses when disturbances arrive simultaneously at applications sharing a slot id according to the proposed dynamic scheduling strategy. All applications meet their respective settling time requirements as they switch to the high-cost mode for at least the minimum number of required samples before the maximum waiting time.

impacted by the state of other applications sharing the same slot id. That is, the response of $\mathcal{C}_1$ will also be the same if $\mathcal{C}_3$ and $\mathcal{C}_4$ were not disturbed simultaneously with $\mathcal{C}_1$. This is because the switching instants between the low-cost and the high-cost modes for $\mathcal{C}_1$ depends only on the

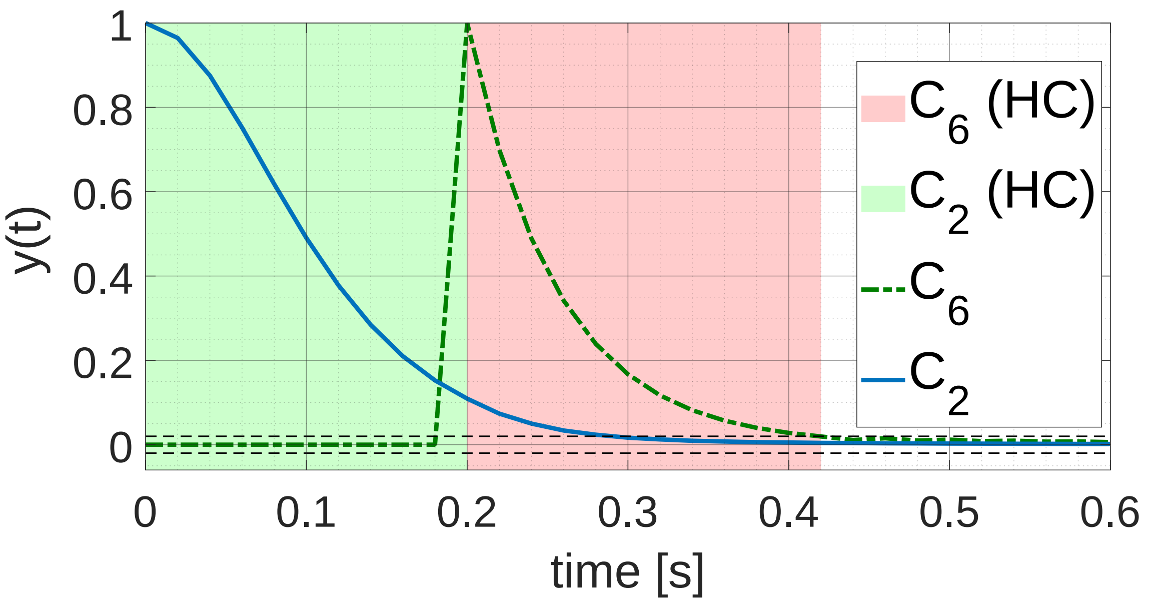


Figure 5.16: Control responses when disturbances arrive separately at applications sharing a slot id according to the proposed dynamic scheduling strategy. Both applications can achieve the maximum performance that is equal to the performance of a controller that operates only in the high-cost mode.

obtained static schedule and is independent of the state of $\mathcal{C}_3$ and $\mathcal{C}_4$. This is the main disadvantage of static scheduling, i.e., even if high-quality resources are available, they cannot be used to improve the performance of an application that is experiencing a disturbance. Therefore, the average control performance of the applications might be lower in case of static scheduling as compared to a dynamic scheduling strategy.

For the proposed dynamic scheduling strategy, we have build two networks of TAs representing two slot ids. Using UPPAAL, we simulate the TA models for the following two cases: (i) Disturbances arrive simultaneously at $\mathcal{C}_1$, $\mathcal{C}_3$, $\mathcal{C}_4$ and $\mathcal{C}_5$. (ii) Disturbance arrives at $\mathcal{C}_6$ 10 samples after the disturbance at $\mathcal{C}_2$. Using the obtained switching sequences from UPPAAL simulations, we simulate the control loops in MATLAB. The response curves for the two cases are shown in Figure 5.15 and Figure 5.16 respectively. The shaded regions indicate the occupants of the slots. It can be verified that each application meets its settling time requirement. $\mathcal{C}_3$ uses s_k for $n^+_{T,3} = 5$ slots as there is no preemption while all others using s_k are preempted at $n^-_{T,i}$. $\mathcal{C}_2$ and $\mathcal{C}_6$ are not preempted and can achieve the maximum performance equal to $J_{HC,2}$ and $J_{HC,6}$ respectively. $\mathcal{C}_2$ stays in the high-cost mode for 10 samples to achieve the settling time of $0.3\,\mathrm{s}$ while the conservative switching in [146, 244] would require $\mathcal{C}_2$ to stay in the high-cost mode for 15 samples and still obtain the same performance.

When disturbance did not arrive simultaneously at $\mathcal{C}_2$ and $\mathcal{C}_6$ sharing the TDMA, they can achieve a settling time of $0.3\,\mathrm{s}$ and $0.22\,\mathrm{s}$ respectively for the proposed dynamic scheduling strategy. However, in case of static scheduling, the settling time of $\mathcal{C}_2$ and $\mathcal{C}_6$ are $0.4\,\mathrm{s}$ and $0.34\,\mathrm{s}$ respectively, and these times would remain the same even if only one of them is disturbed at a time. Thus, the control performance obtained using the proposed dynamic scheduling strategy can be higher on average than that obtained with the proposed static scheduling strategy.

Comments on verification time: Towards solving the resource dimensioning problem for the proposed dynamic scheduling strategy, we have used model checking of networks of TAs. The main drawback of using model checking for schedulability analysis has always been the issue of scalability. In the case study under consideration, all except one verification took less than a minute. Even few took less than a second. A particular case of mapping $\{\mathcal{C}_1, \mathcal{C}_5, \mathcal{C}_4, \mathcal{C}_3\}$ to one slot took close to 5 hours. However, it is possible to accelerate the verification if we do not consider infinite instances of disturbance. For each application, we can calculate the maximum number of disturbance instances in other applications that can coincide with its disturbance. Accordingly, we can adapt the model and verify. With this approach, we can verify the mapping of $\{\mathcal{C}_1, \mathcal{C}_5, \mathcal{C}_4, \mathcal{C}_3\}$ to one slot within 15 minutes (i.e., a speed up of 20 times). We used a computer with Intel(R) Core(TM) i7 $-$ 5600U CPU @2.60 GHz processor and 8 GB RAM for the verification. Note that for the problem setting we study here, it may not be required to analyze too many applications in one slot. Moreover, this whole process being offline, time is not the main constraint here.

5.7 Related Works

Communication-aware design of distributed CPSs [245] has emerged as an important research topic and attracted extensive attention since [246]. The main idea is to consider the characteristics of the underlying communication protocol (e.g., scheduling policy [161, 195, 201] and uncertainties [116, 117]) while designing control algorithms and, accordingly, scheduling control tasks to guarantee safety and performance.

Our work follows the bimodal control strategy proposed in [144], that exploits the hybrid communication protocol. The control scheme suggests that there is no performance degradation when the controlled plant is in steady state and low-quality resources are used. However, using high-quality resources during the transient state can improve the control performance significantly. It has been also shown in [144] that there exists a good trade-off opportunity between the control performance and the amount of high-quality resources being used. Later, there have been efforts in [145, 146, 244] towards dynamic allocation of high-quality resources and correspondingly, minimizing the amount of high-quality resources required to guarantee performance of all applications.

Although these works have also studied FlexRay as the heterogeneous communication resource, none of them consider the fact that resource allocation cannot be reconfigured online in FlexRay. Thus, these works cannot be applied to a CPS setting where the platform does not support dynamic reconfiguration. In this context, we propose a cost-efficient strategy that statically allocate high-quality resources to the bimodal controllers. Moreover, our proposed strategy saves more resources than the existing dynamic scheduling strategies.

In the context of dynamic allocation of high-quality resources, there are two main shortcomings of these works. First, the switching strategy that these works consider is very conservative. [146, 244] have studied a switching scheme where once a controller gets the high-quality resources, it switches back to the low-quality resource only when the plant has reached the steady state. In such a switching strategy, when an application gets the high-quality resource immediately after a disturbance has arrived, it will use only the high-quality resources in the transient state even if another application is disturbed in the meanwhile. This is restrictive because after using the high-quality resources for a certain number of control instances, switching back to the low-quality resources might not violate the requirements. Now, on the other hand, although [145] allows to switch back from the high-quality resources to the low-quality resources in the transient-state, this involves a retransmission cost that is even more pessimistic. Compared to these existing strategies, we study the closed-loop dynamics more accurately and allow to switch back from the high-cost mode to the low-cost mode even in the presence of a disturbance only when we can guarantee that such a mode change will not violate the requirements. This reduces the conservatism involved in the existing strategies. Second, although the switching control dynamics are analyzed in the existing techniques, the full information is not used while dimensioning the high-quality resources. Instead, these works only consider the timing information for the worst switching instant for each application. This leads to over-provisioning of resources. Note that in the critical instant when all applications sharing the high-quality resources experience disturbance at the same time, only one application may encounter the worst switching instant where it uses the high-quality resources for the maximum number of samples, while all other application will partially reject the disturbance using the low-quality resources and therefore, will take less samples of the high-quality resources.

In order to guarantee the worst-case performances of all the applications that are assigned the same slot id, we formulate a network of timed-automata representing the system and use model checking for the verification. This is more accurate and, therefore, less conservative compared to the analytical techniques use in the existing works. Earlier works [145, 146, 244] have adapted the timing analysis technique used for CAN communication [106]. In CAN communication, there is a constant transmission time for each message, while in our case, the number of control instances an application stays in the high-cost mode depends on the number of samples it has waited in the low-cost mode after the disturbance has arrived. Thus, existing works have applied the CAN timing analysis to calculate the maximum number of samples that an application might have to wait for the high-quality resources by assuming that the high-priority applications will always stay in the high-cost mode for the maximum number of samples. This adds a lot of pessimism to the analysis and is, therefore, not accurate.

It may be noted here that the slot sharing among multiple applications requires the underlying communication architecture to be runtime reconfigurable. This is not the case with FlexRay. However, this can be partially addressed by developing a communication middleware as proposed in [237]. The proposed middleware can dynamically allocate a slot to different messages, however, these messages are sent from the same sender ECU. Moreover, it is envisioned that future automotive architectures must be reconfigurable to support automated or semi-automated driving [247, 248]. We also believe that our proposed dynamic scheduling strategy will find extensive applications in the future as resource-efficient design will become very important to handle the increasing size and complexity [249, 250].

5.8 Conclusion

In this chapter, we have studied a bimodal controller for multi-resource automotive CPSs. Here, the controller can switch between a high-cost mode and a low-cost mode. In the high-cost mode, the controller uses high-quality resources and therefore offers high performance. In the low-cost mode, lower quality resources are used, and therefore only a lower level of performance can be guaranteed. For a system comprising several bimodal controllers sharing high-quality resources, we have proposed a cost-efficient strategy to statically allocate the expensive high-quality resources to these controllers such that their respective control requirements are met. In the proposed strategy, each controller gets only the minimum amount of such expensive resources. We have further developed a dynamic scheduling strategy that can increase the average control performance of the applications as compared to the static scheduling. Towards this, we have proposed a switching scheme for the bimodal controllers and a dynamic scheduling policy for the scheduler. Such a dynamic resource allocation leads to the problem of minimum dimensioning of high-quality resources. Towards an accurate resource dimensioning for the proposed dynamic scheduling policy and switching control strategy, we have proposed a nested two layer optimization technique. In the inner layer, we verify the control requirements for a set of applications using one slot id by modeling the system as a network of TAs and then using model checking for the verification. In the outer layer, we use a first-fit heuristic to map applications to slots. Our experiments show that both static and dynamic scheduling strategies that we propose here are more cost-efficient compared to the state-of-art techniques.

Although our proposed schemes can achieve tighter resource dimensioning, it does not optimize the average control performance. In case of static scheduling, we only consider that a controller stays periodically for consecutive instances in the high-cost mode. However, without increasing the resource usage, a different combination of instances in the high-cost mode might increase the average or the worst-case performance. Also note that we do not guarantee optimality for the sake of scalability of the proposed static scheduling approach. There might be a non-consecutive pattern of instances in high-cost mode that uses less resource while still guaranteeing minimum performance. Thus, there might be a scope for improvement in control performance and resource usage while statically allocating slots to applications that can be explored in the future. In case of dynamic scheduling, we force an application to switch to the low-cost mode as soon as they have stayed for the minimum mandatory samples in the high-cost mode, when there is an application waiting for the slot. However, in certain cases, delaying the change might improve the performance of the current occupant of the high quality resource without degrading the performance of the waiting applications. In the future, we can investigate if machine learning techniques can improve the decision making while still guaranteeing the minimum control performance. Furthermore, in case of dynamic scheduling, we have considered only many-to-one mapping from applications to slots as it is more resource-efficient than the many-to-many mapping. However, many-to-many mapping can lead to an increase in the average control performance. Thus, in the future, we can have a partitioned many-to-many mapping, i.e., we can partition the pool of applications into sets where each set of application can use a number of slots if that will not increase the resource usage. For example, using our proposed technique, if we use two slots for six applications, and now, if we allow each of the six applications to use any of the two slots and they still meet the requirements, then such a mapping will offer higher average control performance.

6 Energy- and Time-Optimal Active Cell Balancing[1]

6.1 Introduction

In this thesis, until now, we have studied the CPS problem of designing and implementing distributed embedded control systems, where only the generic feedback control of physical systems is considerd. In this chapter, we study a more specific CPS problem of active cell balancing in high-power battery packs. Here, we discuss the optimal control problem for active cell balancing and propose customized techniques to synthesize the optimal control of the cell balancing process such that the energy dissipation and the balancing time are minimized respectively.

Cell balancing in battery packs: High-power battery packs, consisting of multiple individual series-connected cells, are gaining more importance, especially with the rapid introduction of EVs and HEVs. Due to manufacturing variances and nonuniform temperature distribution within the pack, charge and discharge rates of the individual cells may differ, thereby resulting in variations in their charge levels (as discussed in Section 1.1.5). The cell with the lowest charge level determines the discharging threshold, and similarly, the cell with the highest charge will cut-off the charging process. This premature cut-off of the charging and the discharging processes significantly reduces the usable capacity of the battery pack. Cell balancing is typically performed to minimize the charge variations in a battery pack. Conventional approaches for cell balancing are *passive*, where the excess charge in cells is dissipated as heat across a high power resistor, and are therefore energy-inefficient [163]. By contrast, energy-efficient *active* cell balancing approaches are an emerging alternative where the excess charge is redistributed

[1]This chapter is compiled based on two publications as follows: (i) "Multi-stage optimization for energy-efficient active cell balancing in battery packs" [251] appeared at the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). (ii) "Optimal scheduling for active cell balancing" [252] appeared at the 2019 IEEE Real-Time Systems Symposium (RTSS).

across the cells. Note that this redistribution is not lossless due to the parasitic resistances and capacitances of the circuit components in the balancing architecture. Moreover, active cell balancing is typically performed using a very low current due to certain physical limitations of the charge transfer circuit architectures, and therefore, a balancing session for a battery pack requires a significant amount of time.

Balancing goals: Minimizing the energy dissipation and the time taken to equalize the charge levels of the cells in a pack are the important optimization objectives for cell balancing [82–84]. Lower energy dissipation for cell balancing during the charging process implies lower cost. Moreover, cell balancing can also be performed on a car to increase the driving range in the absence of a charging source. In such cases, lower energy dissipation directly correlates to higher charge and, practically, a higher driving range. When cell balancing is performed during the charging process of the battery pack, a prolonged balancing time will eventually increase the charging time of the pack. For instance, the charging process is stopped as soon any cell, in a series-connected pack, is fully charged irrespective of the charge levels on the other cells. The cell balancing process is then initiated for the charge equalization. Once the charge levels of all the cells are equalized, the battery pack is not fully charged anymore and again the charging process is initiated. Therefore, minimizing the cell balancing time (interchangeably also called as the charge equalization time) significantly reduces the charging time, thereby improving the user acceptance in case of EVs. Similarly, when the car battery is dying during driving, quickly increasing the charge level of the weakest cell will extend the driving range so that the nearest charging station can be reached.

Balancing architectures and algorithms: Until now, active cell balancing has been studied in the power electronics domain, where different hardware circuit architectures have been proposed for charge transfers, see [85, 164, 165, 253–257]. Charge transfer between cells is facilitated using temporary energy storage elements such as inductors, capacitors or transformers coupled with a power MOSFET switching network. Several balancing architectures with varying charge transfer features such as balancing between adjacent cells [164], direct charge transfer between non-adjacent cells [85] and charge transfers between groups of cells [165] have been proposed in the literature. Although these architectures have progressively improved the cell balancing process in terms of energy dissipation and balancing time, the full potential of these architectures has not been explored yet.

Besides the balancing architectures, software algorithms that control the charge transfer significantly influence the energy dissipation and the total time of an equalization process. The control algorithm here is basically a feedback scheduler and its main task is to schedule charge transfers optimally based on the SoCs of the cells in the pack. Different algorithms for controlling the cell balancing architectures came up in the recent past [84, 165, 255, 256, 258, 259]. These algorithms adopted an iterative approach either based on heuristics or by borrowing concepts from control theory to select charge transfer pairs of cells for minimizing the overall charge variation. However, these techniques only guaranteed charge equalization. No optimality results were known until our works on cell balancing that we will describe in this chapter.

Challenges towards optimal cell balancing: Formulating a scalable one-step optimization problem to synthesize the optimal charge transfer schedule is non-trivial. This is mainly due to the constraints imposed by the underlying balancing architecture for selecting the charge trans-

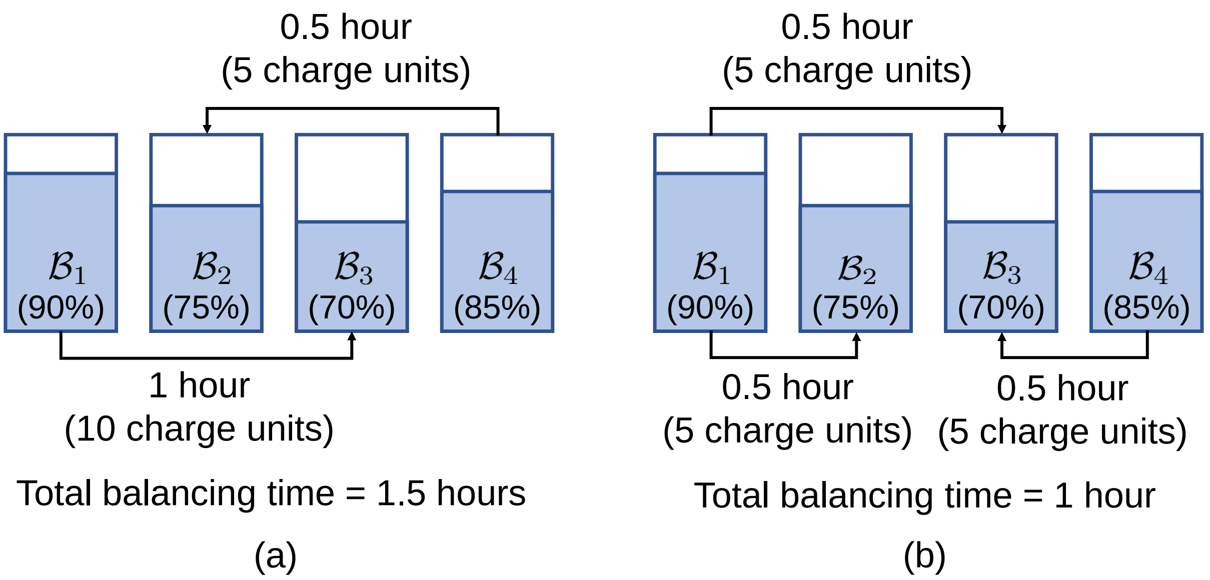


Figure 6.1: Cell balancing in a battery pack of four cells. There are two balancing options. (a) $\mathcal{B}_1$ transfers charge to $\mathcal{B}_3$ and then $\mathcal{B}_4$ to $\mathcal{B}_2$. (b) $\mathcal{B}_1$ and $\mathcal{B}_2$ transfers charge in parallel with $\mathcal{B}_3$ and $\mathcal{B}_4$, and then $\mathcal{B}_1$ and $\mathcal{B}_3$ equalizes.

fer pairs of cells. For instance, no two charge transfer pairs can share the same current flow path at the same time, as it will result in a short circuit condition. For example, in the non-neighbor charge transfer circuit architecture, as explained in Section 2.3.1 and illustrated in Figure 2.6, charge transfer between the cells $\mathcal{B}_1$ and $\mathcal{B}_3$ blocks the cell $\mathcal{B}_2$ to get involved in charge transfer simultaneously. Such scheduling constraints will be further discussed in Section 6.3.2.1. Furthermore, not all cells can be paired with each other since the maximum range in which a cell can transfer charge in a series connection is limited by the capabilities of the underlying balancing architecture. A more detailed discussion on this constraint is provided in Section 6.2.1. We also know that charge transfer occurs in cycles as described in Section 2.3.2.

The charge transfer schedule is composed of time slots, where in each slot, multiple charge transfers can be scheduled that do not cause any short circuit condition. Here, the length of each time slot is, therefore, equal to that of a charge transfer cycle assuming that the operation frequency of all MOSFET switches in a charge transfer circuit is a constant. Now, considering the aforementioned constraints, a closed-form problem formulation will require a binary variable representing the schedule of a feasible charge transfer pair of cells in a time slot. A cell balancing session will typically comprise hundreds of millions of such time slots, and therefore, such a problem formulation cannot be feasibly solved.

Minimizing the overall balancing time of a cell balancing session essentially boils down to minimizing the number of time slots in the balancing schedule. This is even more challenging because the balancing time depends non-trivially on the duration for which a pair of cells performs charge transfer. This is mainly because of the scheduling constraints imposed by the balancing architecture. For the example in Figure 6.1, there are two balancing options, as shown in Figure 6.1(a) and Figure 6.1(b) respectively. Both options would ensure charge equalization, however, the charge transfers in Figure 6.1(b) will result in a lower balancing time. In Figure 6.1(a), the pairs $\pi_{1,3}$ (i.e., $\mathcal{B}_1$ and $\mathcal{B}_3$ are the source and destination cells respectively) and $\pi_{4,2}$ cannot be simultaneously scheduled, and therefore, the overall balancing time is the sum of the individual times (i.e., $1\,\mathrm{h}$ + $0.5\,\mathrm{h}$ = $1.5\,\mathrm{h}$). For the option in Figure 6.1b, $\pi_{1,2}$ and $\pi_{4,3}$ can transfer charge in parallel while $\pi_{1,3}$ has scheduling conflicts with $\pi_{1,2}$ and $\pi_{4,3}$. Let us denote the time taken by $\pi_{1,2}$, $\pi_{4,3}$ and $\pi_{1,3}$ as x, y and z respectively. Thus, exploiting the capability of the balancing architecture to support concurrent charge transfers, the overall balancing time for this option is given by: $\max(x, y) + z = \max(0.5\,\mathrm{h}, 0.5\,\mathrm{h}) + 0.5\,\mathrm{h} = 1\,\mathrm{h}$. Note that for the two balancing options in Figure 6.1, the times for charge equalization are calculated using two different models. Moreover, considering that the concurrency relation between the pairs is intransitive, it is not trivial to obtain a closed form expression for the total balancing time for a large battery pack (e.g., 96 cells) consisting of more than a thousand feasible charge transfer pairs. Furthermore, the model of the balancing time will depend on the nature of these charge transfers, e.g., possibility of preempting a charge transfer once started.

Existing approaches [84, 256] follow an iterative scheduling strategy, where in each step charge transfer pairs of cells are selected while reducing the variance in SoCs of the cells in the pack. However, these approaches are based on heuristics and do not guarantee optimality. For such an approach, formulating an optimization objective in each iteration to minimize both energy dissipation and charge variance, is challenging since these two objectives are conflicting. For example, to minimize the energy dissipation in each step, the obvious solution would be to not perform any charge transfers. However, this will never equalize the SoCs of the cells. Moreover, the charge transfer pairs selected during the initial stage might not be optimal at a later point in time since the SoCs of the selected cells change during the course of balancing. It is difficult to determine the time for which the selected pairs must balance and when to re-evaluate the next set of optimal charge transfer pairs.

By borrowing concepts from control theory, it is possible to guarantee charge equalization (i.e., asymptotic stability) [258,259]. However, the optimal cell balancing problems under study do not fit into any standard optimal control framework due to the architectural constraints that need to be respected.

Contributions: Towards optimal cell balancing, we make the following contributions:

- We propose a two-stage optimization problem formulation to minimize energy dissipation. As such the cell balancing problem does not fit into any standard closed-form optimization framework, we exploit the functional behavior of the underlying charge transfer circuit architectures to partition the whole problem into two stages while preserving the guarantee on minimum energy dissipation.
  - In the first stage, we propose a mixed-integer linear programming (MILP) problem formulation to minimize the energy dissipation of a cell balancing process. We exploit the fact that only the number of charge transfer cycles between each feasible pair of cells have an impact on the energy dissipation, while the exact order in which the charge transfers are performed does not influence the energy dissipation. The output of this first stage will identify the energy-optimal charge transfer pairs $\pi_{i,j}$s and the corresponding number of cycles $c_{i,j}$s for which these pairs must perform charge transfer.
  - We show that the schedule synthesis in the second stage can be formulated as a minimum vertex coloring (MVC) problem. Here, each charge transfer cycle of a selected pair of cells

forms a node in a conflict graph and if an edge connects two nodes then the corresponding two charge transfer cycles cannot be schedule in one slot. Hence, the minimum number of colors required to color all vertices in the conflict graph is equal to the minimum number of time slots necessary for charge equalization.

- We prove that the conflict graph constructed in the second stage can represent an interval graph. Therefore, the conflict graph is also chordal as mentioned in Section 2.4.5. By exploiting specific properties of interval graphs and chordal graphs, we derive an expression for the minimum schedule length in terms of the number of cycles each pair must transfer charge, which is also equivalent to the overall balancing time. Now, we use the derived expression for the overall balancing time as the optimization objective in the MILP problem of the first stage. Hence, we can solve the MILP to directly minimize the balancing time.

- Given that the conflict graph is chordal, we can apply lexicographic breadth first search (Lex-BFS) and greedy vertex coloring algorithms to color the vertices of the graph using minimum number of colors. We propose a memory- and time-efficient implementation of these algorithms to solve the scheduling problem in the second stage for optimizing energy dissipation and balancing time respectively. Here, we exploit the fact that two nodes representing two charge transfer cycles of the same charge transfer pair will always be in the same clique of the conflict graph.

- Experiments show that compared to state-of-the-art approaches, our proposed optimization can reduce the energy dissipation by up to $78.6\,\mathrm{Wh}$ (i.e., from $211.61\,\mathrm{Wh}$ to $133.01\,\mathrm{Wh}$), when the energy dissipation is considered as the optimization objective. On the other hand, when we optimize for the balancing time, we can improve it by up $6.22\,\mathrm{h}$ (i.e., from $11.04\,\mathrm{h}$ to $4.82\,\mathrm{h}$) compared to existing heuristics.

Chapter organization: The rest of this chapter is organized as follows. In Section 6.2, we mathematically model the constraints for charge equalization considering the details of the underlying charge transfer circuit architectures. Section 6.3 describes the proposed two-stage optimization framework for minimizing the energy dissipation, where in the first stage, the optimal charge transfers are obtained that minimize the energy dissipation, while in the second stage, the obtained charge transfers are scheduled. Section 6.4 shows that the same two-stage framework can also be used for minimizing the balancing time. In particular, first a model is derived for the balancing time exploiting concepts of graph theory, and subsequently, it is explained how this model can be used in the first stage of the optimization to determine the optimal charge transfers that minimize the balancing time. Section 6.5 illustrates the customized implementation of the Lex-BFS and the greedy vertex coloring algorithms respectively exploiting specific characteristics of the scheduling problem in the second stage. Section 6.7 mentions the related works and Section 6.8 provides the concluding remarks.

6.2 Constraint Formulation for Charge Equalization

In this section, we mathematically model the constraints imposed by the neighbor and the non-neighbor charge transfer circuits on the charge equalization process. The charge transfer process between a pair of cells using these circuits is described in Section 2.3.1, while the corresponding mathematical models are provided in Section 2.3.2. The constraints derived in this section can also be extended for other charge transfer circuit architectures depending on their charge transfer patterns.

6.2.1 Feasible Pairs of Cells for Charge Transfer

Let us consider a battery pack $\mathbb{B}$ comprising N_B series-connected cells. These cells are denoted as $\{\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_{N_B}\}$ in the order of their serial electrical connection in the pack. Constrained by the voltage rating of the switches used in the balancing architecture, there can be a maximum $\widehat{d}$ number of cells in between a charge transfer pair of cells. For example, when $\mathcal{B}_1$ transfers charge to $\mathcal{B}_4$ in Figure 2.6c, then M_T^1 needs to withstand an approximate terminal voltage of $V_1 + V_2 + V_3$. Thus, switches with higher rating (higher cost) will be required to increase $\widehat{d}$. For neighbor-only balancing architecture, $\widehat{d} = 0$. Corresponding to $\widehat{d}$, a cell $\mathcal{B}_i$ can be paired with any cell belonging to the list $\mathcal{P}_i$, where:

$$\mathcal{P}_i = \{\mathcal{B}_j | j \neq i \ \wedge \ \max(i - \widehat{d} - 1, 1) \leq j \leq \min(i + \widehat{d} + 1, N_B)\}. \tag{6.1}$$

The set of all feasible charge transfer pairs can be written as:

$$\mathcal{CT} = \left\{\pi_{i,j} | (\mathcal{B}_i, \mathcal{B}_j) \in \bigcup_{i=1}^{N_B} \{\mathcal{B}_i\} \times \mathcal{P}_i \right\}. \tag{6.2}$$

Note that $A \times B$ implies Cartesian product between two sets A and B. Here, $\pi_{i,j}$ denote an ordered pair of cells $\mathcal{B}_i$ and $\mathcal{B}_j$ where $\mathcal{B}_i$ is the source cell and $\mathcal{B}_j$ is the destination cell. Here, for two cells $\mathcal{B}_a$ and $\mathcal{B}_b$ (where, $0 < |a - b| \leq \widehat{d} + 1$), $\mathcal{B}_a$ can transfer charge to $\mathcal{B}_b$ or $\mathcal{B}_b$ can transfer charge to $\mathcal{B}_a$, i.e., $\pi_{a,b}, \pi_{b,a} \in \mathcal{CT}$. Thus, we do not fix the source and destination cells during modeling and let the solver take the decision based on the optimization problem.

6.2.2 Charge Transfer Cycles

Typically, the voltage of a Li-ion cell changes negligibly throughout the operating region [83, 84]. Moreover, cell balancing typically takes place when the charge levels of the cells are within a very short spectrum. Therefore, the cell voltage can be assumed to remain constant during charge equalization. The assumption of a constant voltage leads to a negligible error (less than 0.5%) in estimating the energy dissipation and the time duration for a charge equalization session.

As discussed in Section 2.3, the charge transfer occurs in cycles. Assuming that the cell voltages remain constant during the whole balancing process, the charge transmitted $Q_{tx}(i,j)$ by the source cell $\mathcal{B}_i$ and the charge received $Q_{rx}(j,i)$ by the destination cell $\mathcal{B}_j$, in one cycle, are constants, according to Eq. (2.48) and Eq. (2.53) respectively. Now, for $\pi_{i,j} \in \mathcal{CT}$, let $c_{i,j}$ be the number of cycles $\mathcal{B}_i$ transfers charge to $\mathcal{B}_j$ in an entire balancing session. In these $c_{i,j}$ cycles, the charge transmitted $Q_{i,j}^-$ by $\mathcal{B}_i$ and the charge received $Q_{j,i}^+$ by $\mathcal{B}_j$ are given by:

$$Q_{i,j}^- = Q_{tx}(i,j) \cdot c_{i,j} \quad \text{and} \quad Q_{j,i}^+ = Q_{rx}(j,i) \cdot c_{i,j}. \tag{6.3}$$

Therefore, the total charge (Q_i^T) transmitted by $\mathcal{B}_i$ and total charge (Q_i^R) received by $\mathcal{B}_i$ during the charge equalization process can be written as:

$$Q_i^T = \sum_{\mathcal{B}_j \in \mathcal{P}_i} Q_{i,j}^- = \sum_{\mathcal{B}_j \in \mathcal{P}_i} Q_{tx}(i,j) \cdot c_{i,j}, \tag{6.4}$$

$$Q_i^R = \sum_{\mathcal{B}_j \in \mathcal{P}_i} Q_{i,j}^+ = \sum_{\mathcal{B}_j \in \mathcal{P}_i} Q_{rx}(i,j) \cdot c_{j,i}. \tag{6.5}$$

Note that $c_{i,j}$ denotes the number of cycles $\mathcal{B}_i$ transfers charge to $\mathcal{B}_j$ while $c_{j,i}$ is the number of cycles $\mathcal{B}_i$ receives charge from $\mathcal{B}_j$. As mentioned earlier, we do not fix the role of a cell in charge transfer while modeling the constraints.

For each cycle of charge transfer from $\mathcal{B}_i$ to $\mathcal{B}_j$, there is a certain amount of energy dissipation. Let V_i and V_j be the cell voltages of $\mathcal{B}_i$ and $\mathcal{B}_j$ respectively. Then, the energy transferred from $\mathcal{B}_i$ and the energy received by $\mathcal{B}_j$ are respectively given by:

$$E_{tx}(i,j) = Q_{tx}(i,j) \cdot V_i \text{ and } E_{rx}(j,i) = Q_{rx}(j,i) \cdot V_j. \tag{6.6}$$

Thus, the energy dissipated per cycle $E_\triangle(i,j)$ as heat in the parasitic resistances of the circuit components is:

$$E_\triangle(i,j) = E_{tx}(i,j) - E_{rx}(j,i) = Q_{tx}(i,j) \cdot V_i - Q_{rx}(j,i) \cdot V_j. \tag{6.7}$$

Here, $E_\triangle(i,j)$ is also a constant for our assumption of constant cell voltage. For $c_{i,j}$ cycles of charge transfer from $\mathcal{B}_i$ to $\mathcal{B}_j$, the energy dissipation $E_{i,j}$ is:

$$E_{i,j} = E_\triangle(i,j) \cdot c_{i,j}. \tag{6.8}$$

Note that this value is independent of the exact schedule for these $c_{i,j}$ cycles.

We assume that the operation frequency of the charge transfer circuit is constant. That is, the duration of a charge transfer cycle is independent of the pair of cells involved in the transfer and is a constant T^C. Thus, $c_{i,j}$ cycles of charge transfer between a pair $\pi_{i,j}$ will require $c_{i,j} \cdot T^C$ time units.

6.2.3 Balanced Battery Pack

It must be ensured that the charge transfers must lead to a balanced battery pack. Here, we formulate two sets of constraints to ensure that all the cells in the battery pack must have their charge levels within a certain range after charge equalization. These constraints are explained as follows.

(i) The first set of constraints considers that the change in charge levels of a cell is equal to the difference in the amounts of charge received and the amounts of charge transmitted by the cell. Let $Q_{i,s}$ and $Q_{i,f}$ be the initial and final charge levels of a cell. The total charge transmitted (Q_i^T) and received (Q_i^R) by a cell are given by Eq. (6.4) and Eq. (6.5) respectively. Thus, the constraints can be written as follows:

$$\forall \mathcal{B}_i \in \mathbb{B}, \quad Q_{i,f} - Q_{i,s} = Q_i^R - Q_i^T = \sum_{\mathcal{B}_j \in \mathcal{P}_i} Q_{rx}(i,j) \cdot c_{j,i} - Q_{tx}(i,j) \cdot c_{i,j}. \tag{C1}$$

Here, the summation over $\mathcal{P}_i$ ensures that a cell can perform a charge transfer only within a certain distance $\widehat{d}$, according to Eq. (6.1). The variables $c_{i,j}$s will ensure that if a transmitting cell loses charge then a receiving cell will gain charge according to Eq. (6.3).

(ii) The second set of constraints considers that the final charge levels of the cells are within a certain threshold Q_{th}. In other words, the difference between the charge levels of any two cells in the battery pack must be less than or equal to Q_{th}. These constraints can be formulated as follows:

$$\forall \mathcal{B}_i, B_j \in \mathbb{B}, \quad -Q_{th} \leq Q_{i,f} - Q_{j,f} \leq Q_{th}. \tag{C2}$$

Note that in Eq. (C1) and Eq. (C2), the only variables are the final charge levels $Q_{i,f}$s and the number of charge transfer cycles between all feasible pairs of cells $c_{i,j}$s. $Q_{i,f}$s are continuous while $c_{i,j}$s are integer variables. Also, note that both constraints are linear.

6.3 Minimizing Energy Dissipation

In this section, we describe the proposed two-stage optimization to determine a charge transfer schedule that minimize the energy dissipation for cell balancing. In the first stage, we formulate an MILP problem to determine the optimal charge transfers that minimize energy dissipation and guarantee charge equalization. In the second stage, we formulate an MVC problem to schedule the obtained charge transfers from the first stage in minimum time.

6.3.1 Stage 1: Determining Optimal Charge Transfers

The energy dissipated $E_{i,j}$ during charge transfer in a cell balancing session between a pair $\pi_{i,j}$ is given by Eq. (6.8). Thus, the total energy dissipation E_D during a charge equalization process is the summation of all the individual losses as follows:

$$E_D = \sum_{\pi_{i,j} \in \mathcal{CT}} E_{i,j} = \sum_{\pi_{i,j} \in \mathcal{CT}} E_{\Delta}(i,j) \cdot c_{i,j}. \tag{C3}$$

Here, E_D is a linear function of $c_{i,j}$s. Note that E_D is independent of the exact schedule of $c_{i,j}$ cycles of charge transfer between the pair $\pi_{i,j}$. Furthermore, in Section 6.2.3, we have derived two sets of linear constraints, i.e., Eq. (C1) and Eq. (C2), for charge equalization. Both sets of constraints are independent of the order of charge transfers and depend only on the number of cycles each feasible pair of cells performs charge transfer. Thus, we can formulate an MILP to minimize the energy dissipation while ensuring charge equalization using the variables $Q_{i,f}$s, $c_{i,j}$s and E_D, which can be written as:

$$\textbf{Minimize} \ E_D, \quad \textbf{s.t.} \ \text{(C1), (C2), (C3).} \tag{\textit{OPT-ED}}$$

Solving the MILP problem will determine the minimum possible energy dissipation. The solution will also give the number of cycles each feasible pair of cells must be involved in charge transfer so that the minimum energy dissipation is achieved.

6.3.2 Stage 2: Scheduling Charge Transfers

In addition to the energy dissipation, minimizing the time taken for equalizing the charge levels of the cells is an important objective for cell balancing. Therefore, it is desirable to optimally schedule the charge transfer pairs of cells obtained from Stage 1. Towards this, in this section, we will first formulate the concurrency constraints for charge transfers as enforced by the cell balancing architectures. Subsequently, based on the derived concurrency constraints, we show that the schedule optimization problem can be formulated as an MVC problem. Note that we discuss the method to solve the MVC problem thus formulated in Section 6.5.

6.3.2.1 Concurrent Charge Transfers

For the neighbor and non-neighbor cell balancing architectures under study, concurrent charge transfers are allowed, however, with certain restrictions. For example, in a battery pack of four cells connected in series as $\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4\}$, there can be simultaneous charge transfers between the pairs $\pi_{1,2}$ and $\pi_{3,4}$. Conversely, the pairs $\pi_{1,2}$ and $\pi_{3,2}$ cannot exchange charge at the same time. For non-neighbor architecture, $\pi_{1,4}$ and $\pi_{3,2}$ cannot be scheduled together, although the pairs do not have a common cell.

We must consider concurrency constraints to ensure that two pairs of cells do not use the same circuit path for charge transfer concurrently as this would lead to a short circuit condition. Thus, when $\mathcal{B}_i$ and $\mathcal{B}_j$ exchange charge, then no cell in between them can be involved in charge transfer simultaneously. Moreover, $\mathcal{B}_i$ and $\mathcal{B}_j$ cannot be paired with any other cell for charge transfer at the same time. Thus, when $\pi_{i,j} \in \mathcal{CT}$ is scheduled for charge transfer, the set of cells that are blocked and, therefore, unavailable to be simultaneously paired for additional charge transfers can be written as follows:

$$\mathcal{U}_{i,j} = \{\mathcal{B}_k \in \mathbb{B} | \min(i,j) \leq k \leq \max(i,j)\}. \tag{6.9}$$

For example, when $\pi_{5,9} \in \mathcal{CT}$ is scheduled, then $\mathcal{U}_{5,9} = \{\mathcal{B}_5, \mathcal{B}_6, \mathcal{B}_7, \mathcal{B}_8, \mathcal{B}_9\}$ is the set of cells that cannot appear in any other pair scheduled concurrently.

Now, we can identify the pairs of cells that have scheduling conflict with each other. We can construct a conflict graph $\mathcal{G}$ showing all scheduling conflicts. $\mathcal{G}$ is an undirected graph where each vertex represents a feasible charge transfer pair and an edge between two vertices denote a scheduling conflict. The set of vertices in $\mathcal{G}$ is, therefore, equal to the set of feasible charge transfer pairs ($\mathcal{CT}$). An edge exists between two vertices representing the pairs $\pi_{a,b}$ and $\pi_{c,d}$ respectively, if they are not simultaneously schedulable. Such an edge can be denoted as an unordered set consisting two pairs, i.e., $\{\pi_{a,b}, \pi_{c,d}\} \in \mathcal{E}_G$, where $\mathcal{E}_G$ is the set of edges in $\mathcal{G}$. Thus, we can identify the edges as follows:

$$\mathcal{U}_{a,b} \cap \mathcal{U}_{c,d} \neq \emptyset \iff \{\pi_{a,b}, \pi_{c,d}\} \in \mathcal{E}_G. \tag{6.10}$$

Example: Figure 6.2 shows an example conflict graph $\mathcal{G}$ for a battery pack with 4 cells in series. For this graph, $\widehat{d} = 0$ is considered for better readability. There are 6 vertices corresponding to 6 feasible charge transfer pairs of cells. Here, $\mathcal{U}_{i,j}$s can be determined as follows:

$$\mathcal{U}_{1,2} = \mathcal{U}_{2,1} = \{\mathcal{B}_1, \mathcal{B}_2\}, \quad \mathcal{U}_{2,3} = \mathcal{U}_{3,2} = \{\mathcal{B}_2, \mathcal{B}_3\},$$
$$\mathcal{U}_{3,4} = \mathcal{U}_{4,3} = \{\mathcal{B}_3, \mathcal{B}_4\}.$$

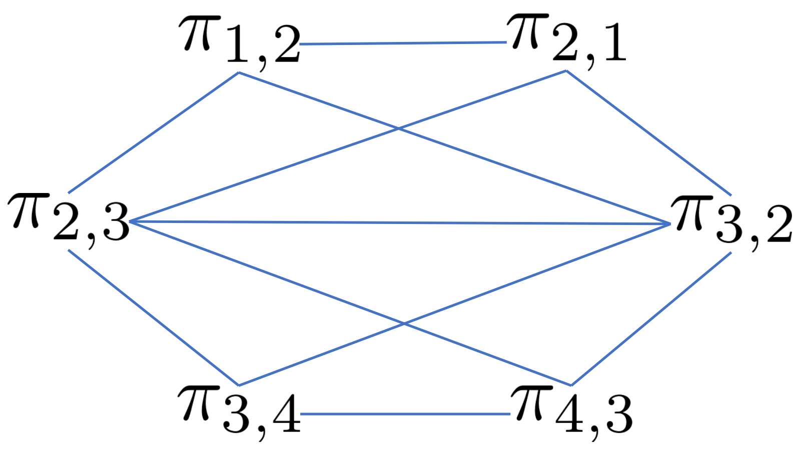


Figure 6.2: An example conflict graph for a battery pack with four cells having a neighbor-only balancing architecture. Each vertex represents a feasible charge transfer pair and an edge between two vertices imply that the corresponding two pairs are not simultaneously schedulable.

We can find all pairs of unconnected vertices as follows:

$$\mathcal{U}_{1,2} \cap \mathcal{U}_{3,4} = \mathcal{U}_{2,1} \cap \mathcal{U}_{3,4} = \mathcal{U}_{1,2} \cap \mathcal{U}_{4,3} = \mathcal{U}_{2,1} \cap \mathcal{U}_{4,3} = \emptyset$$
$$\iff \{\pi_{1,2}, \pi_{3,4}\}, \{\pi_{2,1}, \pi_{3,4}\}, \{\pi_{1,2}, \pi_{4,3}\}, \{\pi_{2,1}, \pi_{4,3}\} \notin \mathcal{E}_G.$$

All other pairs of vertices are connected by edges as shown in Figure 6.2.

6.3.2.2 Equivalent Vertex Coloring Problem for Schedule Synthesis

As mentioned in Section 6.3.1, we obtain the number of charge transfer cycles $c_{i,j}$ for each feasible pair of cells $\pi_{i,j} \in \mathcal{CT}$ from Stage 1. We can annotate the conflict graph $\mathcal{G}$, representing the concurrency constraints, with these values. Thus, a vertex representing a pair of cells $\pi_{i,j} \in \mathcal{CT}$ is annotated with the number of cycles of charge transfer $c_{i,j}$ they must be scheduled for. For the example of four series-connected cells in Figure 6.2, let us assume that:

$$c_{1,2} = 2, \quad c_{2,1} = 0, \quad c_{2,3} = 0, \quad c_{3,2} = 1, \quad c_{3,4} = 2, \quad c_{4,3} = 0.$$

The annotated conflict graph for this example is shown in Figure 6.3(a).

Now, we can reduce the conflict graph by deleting the vertices with $c_{i,j} = 0$, as shown in Figure 6.3(b). We denote the reduced conflict graph as $\mathcal{G}_r$. This is possible because the corresponding pairs of cells have no scheduling requirement, and therefore, it is not required to consider concurrency constraints involving them.

Furthermore, we expand the reduced conflict graph $\mathcal{G}_r$, as shown in Figure 6.3(c), such that each vertex represents one charge transfer cycle of a pair of cells (with non-zero requirement). Thus, this graph not only contains information on the concurrency constraints but also represent the scheduling requirements. This detailed graph is denoted as $\mathcal{G}_d$. Let $\pi_{i,j} \in \mathcal{CT}$ be a charge transfer pair with $c_{i,j} = \zeta > 0$. A vertex representing this pair is now expanded into ζ number of vertices as $\{\pi^1_{i,j}, \pi^2_{i,j}, \cdots, \pi^{\zeta}_{i,j}\}$.

Now, two vertices representing charge transfer cycles of the same pair will have an edge between them, i.e., $\pi^p_{a,b}$ and $\pi^q_{a,b}$ will be connected by an edge. This is because these two charge

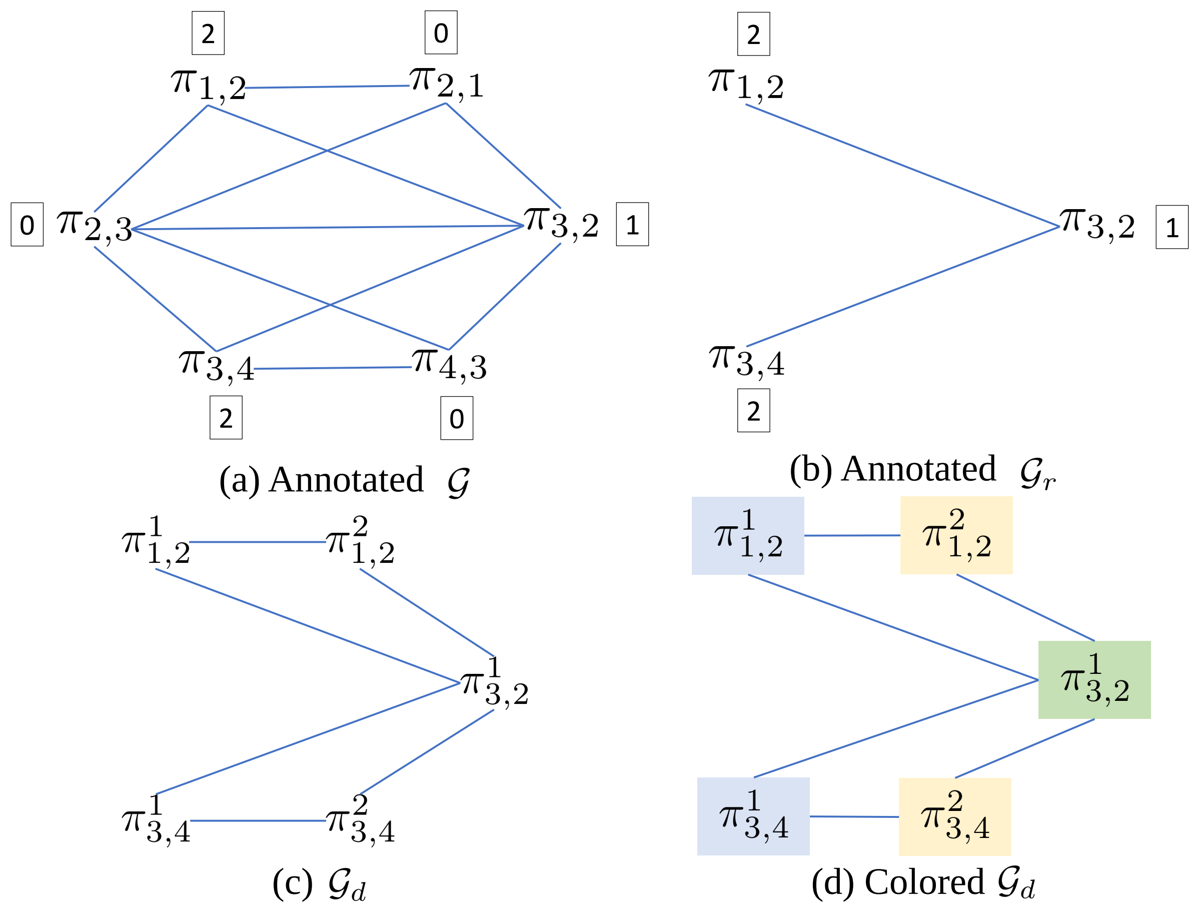


Figure 6.3: Charge transfer schedule synthesis using the minimum vertex coloring (MVC). (a) Annotating the conflict graph with scheduling requirements. (b) Deleting cells pairs with zero requirement. (c) Adding vertex corresponding to each required charge transfer cycle. (d) Synthesizing schedule by optimally coloring the vertices.

transfer cycles cannot be executed in parallel and thus have a scheduling conflict. Note that both $\pi^p_{a,b}$ and $\pi^q_{a,b}$ block the same set of cells, i.e., $\mathcal{U}_{a,b}$, according to Eq. (6.9). Therefore, the intersection set, $\mathcal{U}_{a,b} \cap \mathcal{U}_{a,b} = \mathcal{U}_{a,b}$, is not empty, and it follows from Eq. (6.10) that these two vertices must have an edge in between. Two vertices representing charge transfer cycles of different pairs will have an edge only if the pairs have a scheduling conflict. That is, there is an edge between $\pi^p_{a,b}$ and $\pi^q_{c,d}$ if and only if $\{\pi_{a,b}, \pi_{c,d}\} \in \mathcal{E}_G$ in $\mathcal{G}$.

As defined in Section 2.4.5, vertex coloring is the process of assigning a color to each vertex of a graph such that no two vertices connected by an edge have the same color. The schedule synthesis problem can be formulated as a minimum vertex coloring of the conflict graph $\mathcal{G}_d$ using the following theorem.

Theorem 1. *If the vertices of $\mathcal{G}_d$ can be colored with a minimum of N_G colors then the minimum time T^*_Φ required for scheduling all the charge transfer cycles represented in $\mathcal{G}_d$ is given by*

$$T^*_\Phi = N_G \cdot T^C, \tag{6.11}$$

where, T^C is the duration of a charge transfer cycle determined by the operation frequency of the charge transfer circuit, and it is a constant.

Proof. Let us consider that a charge transfer schedule Φ is composed of equal-sized time slots $\{\phi_1, \phi_2, \cdots, \phi_k, \cdots\}$. Here, each slot is of length equal to T^C. Each charge transfer cycle fits into one slot and several of them can share a slot if they are simultaneously schedulable. Synthesizing a schedule that minimizes the balancing time boils down to determining a mapping of charge transfer cycles to slots such that the minimum number of slots are used.

Now, let us label colors as $\{\omega_1, \omega_2, \cdots, \omega_k, \cdots\}$ and use them to color the vertices of the graph $\mathcal{G}_d$ respecting the rule that two vertices sharing an edge cannot be assigned the same color. In $\mathcal{G}_d$, there is a vertex corresponding to each charge transfer cycle and there exists an edge between two vertices only when the corresponding two charge transfer cycles are not simultaneously schedulable. Thus, we can derive a schedule from the colored graph where the charge transfer cycles corresponding to the vertices colored using ω_k can be mapped to the slot ϕ_k. Now, if N_G is the minimum number of colors used to color all vertices in the graph then it is possible to schedule all charge transfer cycles in N_G slots. The corresponding balancing time is given by $N_G \cdot T^C$ as T^C is the duration of each slot. Thus, the scheduling problem is equivalent to coloring the vertices in $\mathcal{G}_d$ with the minimum number of colors. ■

Example: Figure 6.3(d) shows that three colors are required to color all five vertices in the graph $\mathcal{G}_d$. The total time required to schedule five charge transfer cycles is $3T^C$ instead of $5T^C$. Thus, $2T^C$ time units could be saved by allowing concurrent charge transfers as compared to a sequential schedule, where the charge transfer pairs are scheduled one after the other.

6.4 Minimizing Balancing Time

In the previous section, the two-stage optimization problem formulation can only guarantee minimum energy dissipation. In this section, we will show how a similar framework can also be used to minimize the balancing time. However, balancing time will depend on the exact schedule of the charge transfers. Therefore, we will exploit the characteristics of the MVC problem formulated in the second stage of the optimization to derive a model for the balancing time in terms of $c_{i,j}$s (i.e., the number of cycles for which each feasible pair of cells performs charge transfer). Using the derived model, we will formulate an MILP problem in the first stage to minimize the balancing time. Solving this MILP, we will obtain the values of $c_{i,j}$s that can guarantee minimum balancing time.

6.4.1 Modeling Balancing Time

In Section 6.3.2.2, we have formulated an MVC problem to determine the minimum-length charge transfer schedule for given values of $c_{i,j}$s. Here, we will show that it is possible to optimally solve the MVC problem in linear time for the detailed conflict graph $\mathcal{G}_d$ that is constructed based on the scheduling requirements of the feasible charge transfer pairs and the concurrency constraints between these pairs. In particular, we show that $\mathcal{G}_d$ can represent an interval graph equivalently. Considering that interval graphs are chordal, $\mathcal{G}_d$ is a chordal graph. Therefore, MVC of $\mathcal{G}_d$ is possible in linear time using Lex-BFS and greedy vertex coloring in subsequent steps respectively (as described in Section 2.4.5). Furthermore, we exploit the characteristics of interval graphs, as described in Section 2.4.5, to derive an expression for the minimum number of colors required to color $\mathcal{G}_d$, which is also equivalent to the balancing time, as per Theorem 1.

6.4.1.1 Proof of Chordality

A graph is called chordal if every induced cycle in it, with four or more vertices, has at least one chord. Interval graphs are chordal, where each vertex represents an interval on a real line and there is an edge between two vertices if the corresponding two intervals overlap. Chordal graphs and interval graphs are explained with examples in Section 2.4.5. Here, we prove that the conflict graph $\mathcal{G}_d$, as derived in Sec. 6.3.2.2, is a chordal graph from the definition of interval graphs using Lemma 1.

Lemma 1. *$\mathcal{G}_d$ can represent an interval graph.*

Proof. We prove the lemma by defining an interval $I^r_{i,j}$ for each vertex $\pi^r_{i,j}$ in $\mathcal{G}_d$ as follows:

$$\min(i,j) - \epsilon < I^r_{i,j} < \max(i,j) + \epsilon. \tag{6.12}$$

Here, ϵ is a constant positive real number and $\epsilon < 0.5$. Without loss of generality, we have assumed here open intervals as seen in Eq. (6.12), i.e., the intervals do not include the endpoints.

From the definitions of $\mathcal{U}_{i,j}$ and $I^r_{i,j}$ in Eq. (6.9) and Eq. (6.12) respectively, it can be observed that the integers contained in an interval $I^r_{i,j}$ correspond to the indices of the cells in $\mathcal{U}_{i,j}$. As $\epsilon < 0.5$, two intervals $I^p_{a,b}$ and $I^q_{c,d}$ will overlap if and only if $\mathcal{U}_{a,b}$ and $\mathcal{U}_{c,d}$ have a non-empty intersection set. This can be written as:

$$I^p_{a,b} \cap I^q_{c,d} \neq \emptyset \iff \mathcal{U}_{a,b} \cap \mathcal{U}_{c,d} \neq \emptyset. \tag{6.13}$$

Now, from Eq. (6.10) and Eq. (6.13), we can say that two vertices $\pi^p_{a,b}$ and $\pi^q_{c,d}$ share an edge if and only if the corresponding two intervals overlap, i.e., $I^p_{a,b} \cap I^q_{c,d} \neq \emptyset$. Hence, it is proved that the graph $\mathcal{G}_d$ can represent an interval graph where the intervals are given by Eq. (6.12). ■

Example: Let us assume $\epsilon = 0.1$. For a vertex representing the charge transfer cycle $\pi^2_{4,7}$, $I^2_{4,7} \in \,]3.9, 7.1[$ and $\mathcal{U}_{4,7} = \{\mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6, \mathcal{B}_7\}$. Thus, the only integers in $I^2_{4,7}$ are $\{4, 5, 6, 7\}$ which are also the indices of the cells in $\mathcal{U}_{4,7}$. In Figure 6.3(c), $\pi^1_{3,2}$ and $\pi^1_{3,4}$ represent charge transfer cycles of different pairs and they are connected by an edge. In this case, $I^1_{3,2} \in \,]1.9, 3.1[$ and $I^1_{3,4} \in \,]2.9, 4.1[$ where $I^1_{3,2} \cap I^1_{3,4} = \,]2.9, 3.1[\,\neq \emptyset$. Similarly, $\pi^1_{1,2}$ and $\pi^2_{1,2}$ are charge transfer cycles of the same pair of cells and there is an edge between them. For these two vertices, $I^1_{1,2}, I^2_{1,2} \in \,]0.9, 2.1[$ and $I^1_{1,2} \cap I^2_{1,2} = \,]0.9, 2.1[\,\neq \emptyset$. However, as $I^1_{1,2} = \,]0.9, 2.1[$, $I^1_{3,4} \in \,]2.9, 4.1[$ and $I^1_{1,2} \cap I^1_{3,4} = \emptyset$, the vertices corresponding to the charge transfer cycles $\pi^1_{1,2}$ and $\pi^1_{3,4}$ are not connected by an edge.

6.4.1.2 Identifying Maximal Cliques

As stated in Section 2.4.5, for a chordal graph, the minimum number of colors required to color all the vertices in the graph is given by the cardinality of the clique with the maximum number of vertices. A clique consists of a subset of vertices in a graph where each vertex has an edge with every other vertices. Now, a maximal clique is a clique to which no further vertices can be added to form another clique. One or more of the maximal cliques consist of the maximum number of vertices among all cliques in a conflict graph, and hence, determine the minimum colors required to color the vertices of the graph. More details on cliques and maximal cliques have been provided in Section 2.4.5.

Example: In Figure 6.3(c), there are two maximal cliques in $\mathcal{G}_d$: (i) $\{\pi^1_{1,2}, \pi^2_{1,2}, \pi^1_{3,2}\}$ and (ii) $\{\pi^1_{3,2}, \pi^1_{3,4}, \pi^2_{3,4}\}$. Both maximal cliques have three vertices, and thus, we require three different colors to color all vertices in $\mathcal{G}_d$, as shown in Figure 6.3(d).

Sets of conflicting pairs of cells: Let us denote a set of pairs of cells with non-zero charge transfer cycles as $\mathcal{CT}^+$ which is given by:

$$\mathcal{CT}^+ = \{\pi_{i,j} \in \mathcal{CT} | i, j > 0\}. \tag{6.14}$$

We define a set of conflicting pairs of cells $\mathcal{CP}_k$ as follows:

$$\mathcal{CP}_k = \{\pi_{i,j} \in \mathcal{CT}^+ | \mathcal{B}_k \in \mathcal{U}_{i,j}\}. \tag{6.15}$$

According to Eq. (6.10), two pairs $\pi_{a,b}, \pi_{c,d} \in \mathcal{CP}_k$ cannot be scheduled concurrently as the intersection set, given by $\mathcal{U}_{a,b} \cap \mathcal{U}_{c,d}$, is not empty and has at least the cell $\mathcal{B}_k$. All charge transfer cycles of the pairs of cells in $\mathcal{CP}_k$ are conflicting to each other, and therefore, they form a clique cl_k in $\mathcal{G}_d$. We can therefore determine a set of these sets of conflicting pairs as $\mathcal{S}_{\mathcal{CP}} = \{\mathcal{CP}_1, \mathcal{CP}_2, \cdots, \mathcal{CP}_{N_B}\}$, corresponding to which we get a set of cliques $\mathcal{S}_c = \{cl_1, cl_2, \cdots, cl_N\}$. Note that N_B is the total number of cells in the battery pack. Towards determining the maximum clique using these sets, we can derive the following Lemma.

Lemma 2. *All the maximal cliques in $\mathcal{G}_d$ are in $\mathcal{S}_c$.*

Proof. We exploit the theorem given in [192] to identify all the maximal cliques in an interval graph. This theorem is also described in Section 2.4.5 with examples. Using the theorem, we can prove this lemma as follows:

- According to Lemma 1, $\mathcal{G}_d$ is an interval graph. From the sorted endpoints of all the vertices in $\mathcal{G}_d$, we consider two consecutive endpoints where a left endpoint e_s is followed by a right endpoint e_{s+1}. According to Eq. (6.12), we can write, $e_s = K - \epsilon$ and $e_{s+1} = L + \epsilon$. Here $K, L \in \mathbb{Z}^+$ as K and L are the indices of the cells involved in charge transfer. As $\epsilon < 0.5$, we can deduce,

$$\begin{aligned}
&e_s < e_{s+1} \\
\implies &K - \epsilon < L + \epsilon \quad [\because \text{from definition in Eq. (6.12)}] \\
\implies &K < L + 2 \cdot \epsilon \\
\implies &K \le L \quad [\because (2 \cdot \epsilon < 1) \wedge (K, L \in \mathbb{Z}^+)].
\end{aligned}$$

  We can further write $1 \le K \le L \le N_B$ as K and L are indices of the cells in the pack.

- As $K - \epsilon < K \le L < L + \epsilon$, the interval $]K - \epsilon, L + \epsilon[$ must contain the integer K. Thus, if $]K - \epsilon, L + \epsilon[\in I^r_{i,j}$ then $K \in I^r_{i,j}$.

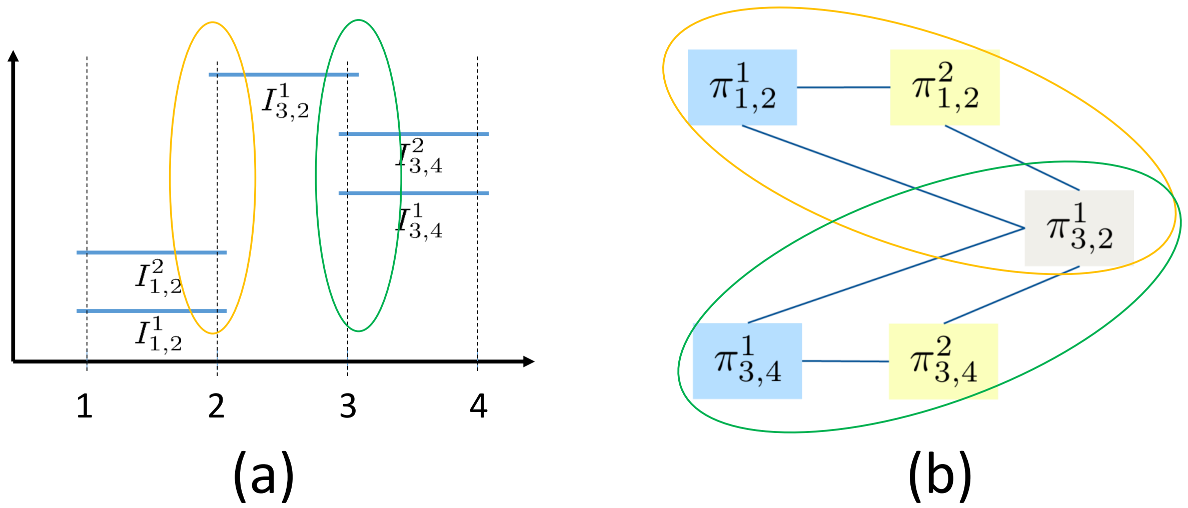


Figure 6.4: Maximal cliques in an interval graph. (a) Five intervals corresponding to the five vertices in Figure 6.3c. The intervals $I^1_{1,2}$, $I^2_{1,2}$ and $I^1_{3,2}$ overlap and the intervals $I^1_{3,2}$, $I^1_{3,4}$ and $I^2_{3,4}$ overlap. (b) Two maximal cliques formed by $\{\pi^1_{1,2}, \pi^2_{1,2}, \pi^1_{3,2}\}$ and $\{\pi^1_{3,2}, \pi^1_{3,4}, \pi^2_{3,4}\}$ respectively.

– As e_s and e_{s+1} are two consecutive endpoints in the sorted list, there are no other endpoints in between them. Thus, all intervals that contain K will also contain $]K-\epsilon, L+\epsilon[$.

– From Eq. (6.9) and Eq. (6.12), we can say that if $K \in I^r_{i,j}$ then $\mathcal{B}_K \in \mathcal{U}_{i,j}$.

– According to Eq. (6.15), each pair of cells $\pi_{i,j} \in \mathcal{CP}_K$ will have $\mathcal{B}_K \in \mathcal{U}_{i,j}$. Thus, the maximal clique formed by the intervals that cover $]K-\epsilon, L+\epsilon[$ is equivalent to the clique c_K formed by the charge transfer cycles of the conflicting pairs of cells in $\mathcal{CP}_K$.

Thus, the maximal clique formed corresponding to the interval $]e_s, e_{s+1}[$ will be in $\mathcal{S}_c$. We can, therefore, conclude that all the maximal cliques of $\mathcal{G}_d$ are in $\mathcal{S}_c$. ■

Example: Let us assume $\epsilon = 0.1$. For the example in Figure 6.3(c), we can sort the endpoints of the intervals corresponding to the five vertices as follows:

$$0.9, 0.9, 1.9, 2.1, 2.1, 2.9, 2.9, 3.1, 4.1, 4.1.$$

The intervals are shown in Figure 6.4(a). Here, 1.9 is a left endpoint and 2.1 is a right endpoint corresponding to the intervals $\pi^1_{3,2}$ and $\pi^1_{1,2}$ respectively. Here, $K = L = 2$ and the open interval $]1.9, 2.1[$ contains the integer $K = 2$. Intervals ($I^1_{1,2}$, $I^2_{1,2}$ and $I^1_{3,2}$) that contain 2 will cover the interval $]1.9, 2.1[$. Now, $\mathcal{CP}_2 = \{\pi_{1,2}, \pi_{3,2}\}$. Thus, the maximal clique formed by the pairs in $\mathcal{CP}_2$ is given by $c_2 = \{\pi^1_{1,2}, \pi^2_{1,2}, \pi^1_{3,2}\}$, as shown in Figure 6.4(b). Similarly, corresponding to the open interval $]2.9, 3.1[$ formed by the left endpoint 2.9 and the right endpoint 3.1 of $I^2_{3,4}$ and $I^1_{3,2}$ respectively, $K = 3$ and $\mathcal{CP}_3 = \{\pi_{3,2}, \pi_{3,4}\}$. Charge transfer pairs in $\mathcal{CP}_3$ will form a clique $c_3 = \{\pi^1_{3,2}, \pi^1_{3,4}, \pi^2_{3,4}\}$, as shown in Figure 6.4(b). Both maximal cliques c_2 and c_3 are in $\mathcal{S}_c$.

Minimum balancing time: Using Theorem 1 and Lemmas 1 and 2, we can derive an expression for the minimum balancing time T^*_Φ as follows.

Theorem 2. *Given the number of charge transfer cycles required by each feasible pair of cells for charge equalization, we can determine the minimum time T^*_Φ required to schedule all the charge transfer cycles as follows:*

$$T^*_\Phi = \max_{c_k \in \mathcal{S}_c} \sum_{\pi_{i,j} \in \mathcal{CP}_k} c_{i,j} \cdot T^C. \tag{6.16}$$

Proof. From Lemma 1, we can say that $\mathcal{G}_d$ is an interval graph and therefore it is chordal. The minimum number of colors N_G required to color $\mathcal{G}_d$ is equal to the cardinality of the maximum clique in $\mathcal{G}_d$. From Lemma 2, we know that all the maximal cliques in $\mathcal{G}_d$ are in $\mathcal{S}_c$. The clique with the maximum cardinality is one of these maximal cliques. Now, let γ_k be the cardinality of the clique c_k. The cardinality of the maximum clique can therefore can be calculated as follows:

$$N_G = \max_{c_k \in \mathcal{S}_c} \gamma_k. \tag{6.17}$$

The clique c_k is formed by the charge transfer cycles of the pairs of cells in $\mathcal{CP}_k$. Therefore, the number of vertices in c_k is equal to the summation of these charge transfer cycles that can be written as follows:

$$\gamma_k = \sum_{\pi_{i,j} \in \mathcal{CP}_k} c_{i,j}. \tag{6.18}$$

Combining Eq. (6.17) and Eq. (6.18), we get

$$N_G = \max_{c_k \in \mathcal{S}_c} \sum_{\pi_{i,j} \in \mathcal{CP}_k} c_{i,j}. \tag{6.19}$$

From Eq. (6.19), using Theorem 1, we get T^*_Φ (as in Eq. (6.16)). ■

Example: For the example in Figure 6.3(c), we can determine the cardinality of the cliques c_ks corresponding to the sets of conflicting pairs as follows:

$$\gamma_1 = c_{1,2} = 2, \quad \gamma_2 = c_{1,2} + c_{3,2} = 3, \quad \gamma_3 = c_{3,2} + c_{3,4} = 3, \quad \gamma_4 = c_{3,4} = 2$$

Therefore, the minimum number of colors required to color the conflict graph $\mathcal{G}_d$ is given by:

$$N_G = \max(\gamma_1, \gamma_2, \gamma_3, \gamma_4) = 3.$$

Both maximal cliques have 3 vertices, i.e., $\gamma_2 = \gamma_3 = 3$, and therefore, $N_G = 3$ as seen in Figure 6.4(b).

6.4.2 Stage 1: MILP formulation

Minimizing balancing time: Until now, we have shown how to determine the balancing time when $c_{i,j}$s are known. Exploiting that, here, we will explain how we can formulate the MILP in the first stage of the optimization framework considering the balancing time as the minimization objective. Towards this, we modify Eq. (6.15) as follows:

$$\mathcal{CP}_k = \{\pi_{i,j} \in \mathcal{CT} | \mathcal{B}_k \in \mathcal{U}_{i,j}\}. \tag{6.20}$$

Thus, we determine sets of conflicting pairs $\mathcal{S}_{\mathcal{CP}} = \{\mathcal{CP}_1, \mathcal{CP}_2, \cdots, \mathcal{CP}_N\}$ based on all feasible cells pairs in $\mathcal{CT}$ irrespective of their required charge transfer cycles. This will not change the clique c_k formed by the pairs of cells in $\mathcal{CP}_k$ because if a pair $\pi_{i,j} \in \mathcal{CP}_k$ has $c_{i,j} = 0$ then there will be no vertices in $\mathcal{G}_d$ corresponding to its charge transfer cycles. The cardinality of the clique, therefore, will not change as can be also observed in Eq. (6.18). That is, if $c_{i,j} = 0$ then it will not add to the value of γ_k. Thus, we can use the expressions in Eq. (6.16) to calculate the balancing time, even with the modified formulation of $\mathcal{CP}_k$ in Eq. (6.20).

Now, we want to determine the values of $c_{i,j}$s such that the balancing time T^*_Φ is minimized. Note that, in Eq. (6.16), T^*_Φ is not a linear function of $c_{i,j}$s. However, γ_ks depend linearly on $c_{i,j}$s. This is a standard minimax optimization and there exists a technique to formulate a linear programming model [260]. In order to formulate a linear objective, we need to consider linear constraints as follows:

$$\forall \mathcal{CP}_k \in \mathcal{S}_{\mathcal{CP}}, \quad \sum_{\pi_{i,j} \in \mathcal{CP}_k} T^C \cdot c_{i,j} \leq T^*_\Phi. \tag{C4}$$

And then we can minimize T^*_Φ.

In summary, the MILP formulation to minimize the balancing time, comprises the variables $Q_{i,f}$s, $c_{i,j}$s and T^*_Φ, and can be written as:

$$\textbf{Minimize} \quad T^*_\phi, \quad \textbf{s.t.} \ \text{(C1), (C2), (C4).} \tag{\textit{OPT-BT}}$$

Minimizing balancing time with energy constraint: In the formulation in Eq. (*OPT-BT*), there is no constraint on energy dissipation. The objectives of balancing time and energy dissipation are often conflicting. That is, to minimize balancing time we might dissipate more heat. For such cases, it is possible to consider a constraint on energy dissipation while minimizing balancing time. Let the bound on energy dissipation be denoted as E^*_D. The constraint on energy dissipation can be written as:

$$E_D \leq E^*_D, , \tag{C5}$$

where, E_D is given by Eq. (C3). Thus, the MILP problem to minimize the balancing time with a budget on the energy dissipation can be formulated as follows:

$$\textbf{Minimize} \quad T^*_\phi, \quad \textbf{s.t.} \ \text{(C1), (C2), (C3), (C4), (C5).} \tag{\textit{OPT-EC-BT}}$$

This formulation can also be used to study the variation of balancing time with energy dissipation. To explore such a trade-off, we can increase E_D gradually starting from the minimum energy dissipation and note the minimum balancing time for each value of E_D by solving the MILP in Eq. (*OPT-EC-BT*).

Minimizing energy dissipation with a deadline on balancing time: The MILP, formulated in Eq. (*OPT-BT*), when solved will determine the minimum possible balancing time for a given initial charge distribution of the battery pack. However, in certain cases, it might be possible to perform cell balancing for a longer time, especially when the EV is parked for a certain known duration. In those cases, we might consider a deadline $\widehat{T}$ on the balancing time based on user feedback and the charge transfer schedule must satisfy the deadline.

In order to ensure that the balancing time is not more than the deadline, we reformulate the constraints in Eq. (C4), and replace T^*_Φ by $\widehat{T}_\Phi$, which is a constant in this case. This can be written as follows:

$$\forall \mathcal{CP}_k \in \mathcal{S}_{\mathcal{CP}}, \quad \sum_{\pi_{i,j} \in \mathcal{CP}_k} T^C \cdot c_{i,j} \leq \widehat{T}_\Phi. \tag{C6}$$

This essentially implies that the charge transfer cycles of a set of conflicting pairs of cells $\mathcal{CP}_k$, given by Eq. (6.20), cannot form a clique c_k in $\mathcal{G}_d$ that has a cardinality greater than $\frac{\widehat{T}_\Phi}{T^C}$. Thus, no clique can have more vertices than the number of colors allowed to color the graph, as determined by $\frac{\widehat{T}_\Phi}{T^C}$. Note that the set of cliques $\mathcal{S}_c$, corresponding to $\mathcal{S}_{\mathcal{CP}}$, contains all the maximal cliques of the graph $\mathcal{G}_d$.

In addition to the constraint on the balancing time, here we consider to minimize the energy dissipation during charge equalization. In summary, we can formulate an MILP to minimize the energy dissipation with a deadline for the balancing time using the variables $Q_{i,f}$s, $c_{i,j}$s and E_D, which can be written as follows:

$$\textbf{Minimize } E_D, \quad \textbf{s.t. } \text{(C1), (C2), (C3), (C6).} \tag{\textit{OPT-DL-ED}}$$

6.5 Efficient Implementation of Minimum Vertex Coloring

By solving the MILP problems in Eq. (*OPT-ED*), Eq. (*OPT-BT*), Eq. (*OPT-EC-BT*) and Eq. (*OPT-DL-ED*), we obtain certain values of $c_{i,j}s$. Corresponding to these values, there is an expected balancing time that can be calculated using Eq. 6.16. In Section 6.4.1, we have derived this expression for balancing time considering that a set of charge transfer cycles $c_{i,j}s$ can be scheduled in minimum time using the minimum vertex coloring algorithm for chordal graphs.

As discussed in Section 2.4.5, vertices of a chordal graph can be colored using the minimum number of colors in two steps. First, a perfect elimination ordering of the vertices (PEOV) of the graph is identified using the lexicographic breadth first search (Lex-BFS) algorithm given in Algorithm 1. In the second step, the greedy vertex coloring algorithm is applied backwards to the obtained PEOV. Here, considering that the colors are numbered as $\{1, 2, \cdots\}$, each vertex v_k is assigned a minimum numbered color that is not used by any of the vertices with which v_k shares an edge.

For a realistic cell balancing scenario, the total number of charge transfer cycles required for charge equalization is in the order of 10^8 to 10^9. The conflict graph $\mathcal{G}_d$, as defined in Section 6.3.2.2, will, therefore, have hundreds of millions of vertices. Applying Lex-BFS and the greedy vertex coloring algorithms on such a huge set of vertices can be cumbersome in terms of memory management and execution. In this section, we propose an efficient implementation of the algorithms using the reduced conflict graph $\mathcal{G}_r$ (defined in Section 6.3.2.2).

The proposed implementation of minimum vertex coloring: In $\mathcal{G}_r$, each vertex represents a pair of cells $\pi_{i,j} \in \mathcal{CT}^+$ and an edge between two vertices determines that the corresponding two pairs cannot be scheduled simultaneously. $\mathcal{G}_d$ is expanded from $\mathcal{G}_r$ where each vertex is a charge transfer cycle $\pi^r_{i,j}$ ($\ni 1 \le r \le c_{i,j}$) of a pair of cells $\pi_{i,j} \in \mathcal{CT}^+$. If two charge transfer cycles cannot be mapped onto the same slot then there is an edge between the corresponding two vertices in $\mathcal{G}_d$. Note that $\mathcal{G}_r$ is chordal which can be proved in a similar way as Lemma 1. Exploiting the relation between $\mathcal{G}_r$ and $\mathcal{G}_d$, we can deduce the following theorem.

Theorem 3. *If $\mathcal{O}_r = (\pi_{a,b}, \cdots, \pi_{k,l}, \cdots, \pi_{y,z})$ is a PEOV for $\mathcal{G}_r$, then $\mathcal{O}_d = (\{\pi^r_{a,b} | 1 \le r \le c_{a,b}\}, \cdots, \{\pi^r_{k,l} | 1 \le r \le c_{k,l}\}, \cdots, \{\pi^r_{y,z} | 1 \le r \le c_{y,z}\})$ is a PEOV for $\mathcal{G}_d$. That is, all the vertices corresponding to the charge transfer cycles of a pair are placed consecutively in $\mathcal{O}_d$ while keeping the relative ordering between the pairs same as in $\mathcal{O}_r$.*

Proof. $\mathcal{CT}^+$, as obtained using Eq. (6.14), is a subset of $\mathcal{CT}$ and comprises the pairs of cells where each pair requires non-zero charge transfer cycles. Each vertex in $\mathcal{G}_r$ represents a cell pair in $\mathcal{CT}^+$. Corresponding to $\mathcal{G}_r$, there is an equivalent subgraph ($\mathcal{G}^1_d$) in $\mathcal{G}_d$ consisting of one charge transfer cycle of each pair in $\mathcal{CT}^+$. Here, the charge transfer cycle $\pi^1_{i,j}$ in $\mathcal{G}^1_d$ corresponds to $\pi_{i,j}$ in $\mathcal{G}_r$. Now, if $(\pi_{a,b}, \pi_{c,d}, \cdots, \pi_{i,j}, \cdots)$ is a PEOV in $\mathcal{G}_r$ then $\mathcal{O}^1_d = (\pi^1_{a,b}, \pi^1_{c,d}, \cdots, \pi^1_{i,j}, \cdots)$ is a PEOV in $\mathcal{G}^1_d$.

Now, if we add one more vertex to $\mathcal{G}^1_d$ corresponding to the charge transfer cycle $\pi^2_{c,d}$, we get another subgraph of $\mathcal{G}_d$ which we denote as $\mathcal{G}^2_d$. For $\mathcal{G}^2_d$, let us consider an ordering of vertices as $\mathcal{O}^2_d = (\pi^1_{a,b}, \pi^2_{c,d}, \pi^1_{c,d}, \cdots, \pi^1_{i,j}, \cdots)$, i.e., $\pi^2_{c,d}$ is placed immediately before $\pi^1_{c,d}$ while keeping the relative ordering of other vertices the same as $\mathcal{O}^1_d$. Each vertex, $\pi^1_{i,j}$ following the vertex $\pi^2_{c,d}$ in $\mathcal{O}^2_d$, is simplicial in the subgraph induced by it and the vertices following it in $\mathcal{O}^2_d$. This is because the induced subgraph remains the same as in case of $\mathcal{O}^1_d$ and $\mathcal{O}^1_d$ is a PEOV in $\mathcal{G}^1_d$. Therefore, $\pi^1_{c,d}$ forms a clique with all its adjacent[2] vertices in the subgraph induced by it and and the vertices following it in $\mathcal{O}^2_d$. Now, $\pi^1_{c,d}$ is adjacent to $\pi^2_{c,d}$ and all adjacent vertices of $\pi^1_{c,d}$ are also adjacent to $\pi^2_{c,d}$ (from the construction of $\mathcal{G}_d$). This implies that there does not exist any vertex following $\pi^1_{c,d}$ in $\mathcal{O}^2_d$ that is adjacent to $\pi^2_{c,d}$ but not adjacent to $\pi^1_{c,d}$. Thus, $\pi^2_{c,d}$ joins the clique formed by $\pi^1_{c,d}$ and its adjacent vertices in the subgraph induced by it and the vertices following it in $\mathcal{O}^2_d$. This implies that $\pi^2_{c,d}$ is simplicial in this induced subgraph.

Now, let us consider a vertex $\pi^1_{a,b}$ preceding $\pi^2_{c,d}$ in $\mathcal{O}^2_d$. $\pi^1_{a,b}$ and its adjacent vertices form a clique in the subgraph induced by it and the vertices following it in $\mathcal{O}^1_d$ as $\mathcal{O}^1_d$ is a PEOV in $\mathcal{G}^1_d$. If $\pi^1_{c,d}$ is not an adjacent vertex of $\pi^1_{a,b}$ then $\pi^2_{c,d}$ also cannot be adjacent to $\pi^1_{a,b}$. Therefore, the same clique will be formed, as in case of $\mathcal{O}^1_d$, in the subgraph induced by $\pi^1_{a,b}$ and the vertices following it in $\mathcal{O}^2_d$. On the other hand, if $\pi^1_{c,d}$ is a vertex in the clique formed by $\pi^1_{a,b}$ and the vertices following it in $\mathcal{O}^1_d$, then $\pi^2_{c,d}$ will join the clique. This is because each adjacent vertex of $\pi^1_{c,d}$ is also adjacent to $\pi^2_{c,d}$. Thus, $\pi^1_{a,b}$ remains simplicial in the induced subgraph formed by it and the vertices following it in $\mathcal{O}^2_d$.

We can add charge transfer cycles of $\pi_{c,d}$ one after the other to form a new subgraph of $\mathcal{G}_d$ and place it in in the ordered set immediately before the last charge transfer cycle of $\pi_{c,d}$. That is, if $\pi^k_{c,d}$ is added to form a subgraph $\mathcal{G}^k_d$ then it is placed immediately before $\pi^{k-1}_{c,d}$ in the new ordered set of vertices $\mathcal{O}^k_d$. $\mathcal{O}^k_d$ is a PEOV in $\mathcal{G}^k_d$ using the same philosophy as explained above. Similarly, we can add the charge transfer cycles of all charge transfer pairs of cells in $\mathcal{CT}^+$ consecutive to each other to form a PEOV in $\mathcal{G}_d$. Given the PEOV $\mathcal{O}_r$ in $\mathcal{G}_r$ we can derive the PEOV $\mathcal{O}_d$ in $\mathcal{G}_d$ by placing the charge transfer cycles of each pair consecutively while

[2]Two vertices are adjacent to each other if they are connected by an edge.

preserving the relative ordering of pairs from $\mathcal{O}_r$. Therefore, $\mathcal{O}_d = (\pi_{a,b}^{c_{a,b}}, \cdots, \pi_{a,b}^{1}, \pi_{c,d}^{c_{c,d}}, \cdots, \pi_{c,d}^{1}, \cdots, \pi_{i,j}^{c_{i,j}}, \cdots, \pi_{i,j}^{1}, \cdots, \pi_{y,z}^{c_{y,z}}, \cdots, \pi_{y,z}^{1})$. Hence, proved. ■

According to Theorem 3, we can, therefore, first apply the Lex-BFS algorithm (i.e., Algorithm 1) to determine a PEOV ($\mathcal{O}_r$) in $\mathcal{G}_r$. Then, we apply the adapted greedy heuristic for vertex coloring, as outlined in Algorithm 7. The main adaptation is that we assign ranges of colors to a vertex in $\mathcal{G}_r$ in each iteration instead of assigning a color to each vertex in $\mathcal{G}_d$. This is possible because we know from Theorem 3 that, a PEOV for $\mathcal{G}_d$ will have the vertices corresponding to the charge transfer cycles of a pair of cells in succession, while keeping the relative ordering of pairs same as $\mathcal{O}_r$.

Algorithm 7: Greedy vertex coloring algorithm.

```
    Input   : $c_{i,j}$s, $\overline{\mathcal{O}}_r$
    Output  : $\Omega$
 1  $\Omega$ = NULL;
 2  for $k \leftarrow 1$ to $|\overline{\mathcal{O}}_r|$ do
 3      $\pi_{a,b} = \overline{\mathcal{O}}_r[k]$;
 4      $\overline{\sigma}$ = NULL;
 5      for $m \leftarrow 1$ to $k-1$ do
 6          $\pi_{c,d} = \overline{\mathcal{O}}_r[m]$;
 7          if $\mathcal{U}_{a,b} \cap \mathcal{U}_{c,d} \neq \emptyset$ then
 8              $\overline{\sigma}$.Append($\Omega[m]$);
 9          end
10      end
11      $\overline{\sigma}_{ord}$ = SortAscending( $\overline{\sigma}$ );
12      $\Omega[k]$ = FindFirstAvailableSlots($\overline{\sigma}_{ord}, c_{a,b}$);
13  end
```

As the greedy vertex coloring algorithm is applied backwards to the PEOV, we take as input $\overline{\mathcal{O}}_r$, that is the reverse ordering of the vertices in $\mathcal{O}_r$. The number of required charge transfer cycles $c_{i,j}$ for each pair of cells $\pi_{i,j} \in \mathcal{CT}^+$ is also taken as input. The output is the mapping σ where, the k-th pair in $\overline{\mathcal{O}}_r$ is mapped to the ranges of slots numbers in $\sigma[k]$, that the pair can use for charge transfer. In line 1, σ is initialized as an empty set. Now, we iterate in order through each pair in $\overline{\mathcal{O}}_r$, and in each iteration, we assign the lowest numbered slots to the current pair, that are not already mapped to any conflicting pair preceding it (lines 2 to 13).

We get the current pair from $\overline{\mathcal{O}}_r$ as $\pi_{a,b}$ (line 3). We use $\overline{\sigma}$ to store the ranges of slots that cannot be assigned to $\pi_{a,b}$, and it is initialized as an empty set (line 4). Now, each pair of cells, with allocated slots, is traversed one by one (lines 5 to 10). In line 6, we take a pair $\pi_{c,d}$ that is preceding $\pi_{a,b}$ in $\overline{\mathcal{O}}_r$. If $\pi_{c,d}$ has a scheduling conflict with the current pair $\pi_{a,b}$ then the ranges of slots used by $\pi_{c,d}$ are appended to $\overline{\sigma}$ (lines 7 to 9). The ranges of unusable slots are then arranged in ascending order and stored in $\overline{\sigma}_{ord}$ (line 11). Note that the ranges cannot be overlapping, because all the preceding vertices that are adjacent to $\pi_{a,b}$ form a clique and therefore are conflicting to each other. This is because $\mathcal{O}_r$ is a PEOV for $\mathcal{G}_r$. In line 12, we search for a set of contiguous slot numbers that can be assigned to $\pi_{a,b}$, avoiding the ranges of slot numbers in $\overline{\sigma}_{ord}$, such that the assigned slots add up to $c_{a,b}$.

At the end of Algorithm 7, we obtain for each pair of cells, with non-zero requirement of charge transfer cycles, the time slots where it must be scheduled. Or in other words, we know the set of pairs that must be scheduled in a time slot. The BMS can take this information and schedule charge transfers accordingly for the battery pack to achieve charge equalization.

6.6 Experimental Results

In this section, we compare our proposed optimization approach to a state-of-the-art heuristic with respect to balancing time and energy dissipation. We also compare the performances of different charge transfer circuit architectures that are studied in this thesis. We further study the impact of the balancing range, peak balancing current and balancing threshold on the charge equalization process.

Simulation setup: We use a realistic EV battery pack of capacity $21.6\,\mathrm{kWh}$ for our case study. This battery pack is obtained by connecting 96 series-connected modules, where each module is made of 24 parallel-connected SAMSUNG INR 18650-25R Li-Ion cells that have a nominal capacity of $2.5\,\mathrm{Ah}$. For this pack, we synthesize different balancing scenarios by generating 60 random initial charge distributions. We take $mn \in \{30, 50, 70\}\%$ and $sd \in \{1, 2, 3, 4\}\%$, where for each combination of mn and sd, five different initial charge distributions are randomly generated with mean mn and standard deviation sd. We consider $Q_{th} = 300\,\mathrm{C}$, i.e., balancing is performed till the charge levels of the cells are within a range of $300\,\mathrm{C}$.

For active cell balancing architectures, commercially available inductors, transformers and MOSFETs are used for our simulations. Accordingly, architectural parameters are:

$$\begin{bmatrix} \mathrm{R_B} & \mathrm{R_L} & R_\mathrm{M} \\ \mathrm{L} & I_\mathrm{peak} & \widehat{d} \end{bmatrix} = \begin{bmatrix} \frac{1}{24}22.5\mathrm{m\Omega} & 12\,\mathrm{m\Omega} & 8.5\,\mathrm{m\Omega} \\ 10\,\mathrm{\mu H} & 12\,\mathrm{A} & 6 \end{bmatrix}.$$

We consider both neighbor-only (*NO*) and non-neighbor (*NN*) balancing architectures respectively for each balancing scenario.

Balancing algorithms: We use the proposed optimization framework to develop four Pareto-optimal schedulers as follows:

- In the *MIN-ED* algorithm, we first minimize the energy dissipation for a charge equalization session using Eq. (*OPT-ED*). We consider the obtained minima as an upper bound to the energy dissipation and correspondingly, minimize the balancing time using Eq. (*OPT-EC-BT*) to obtain a Pareto-optimal set of charge transfers that can be scheduled using the minimum vertex coloring algorithm. The obtained values of energy dissipation and balancing time are denoted as $E_D(1)$ and $T^*_\Phi(1)$ respectively. Note that there does not exist a Pareto-optimal schedule that will have a balancing time higher than $T^*_\Phi(1)$.

- *MIN-BT* algorithm first minimizes the balancing time using Eq. (*OPT-BT*). Then, it considers a time deadline equal to the obtained minima and minimizes the energy dissipation using

Eq. (*OPT-DL-ED*) to get a Pareto-optimal result. Here, we denote the energy dissipation and the balancing time as $E_D(2)$ and $T^*_\Phi(2)$. Note that a Pareto-optimal charge transfer schedule cannot result in a higher energy dissipation than $E_D(2)$.

- Note that $E_D(1)$ and $E_D(2)$ are the minimum and the maximum energy dissipation a Pareto-optimal balancing schedule can offer. In *HALF-ED*, we take the average of $E_D(1)$ and $E_D(2)$ and consider as an upper bound to the energy dissipation to minimize the balancing time using Eq. (*OPT-EC-BT*). We consider the obtained balancing time as a deadline and minimize the energy dissipation using Eq. (*OPT-DL-ED*) to obtain a Pareto-optimal result. The Pareto-optimal values of energy dissipation and balancing time are denoted as $E_D(3)$ and $T^*_\Phi(3)$.
- $T^*_\Phi(1)$ and $T_\Phi(2)$ are the maximum and the minimum balancing time for a Pareto-optimal solution. In *HALF-BT*, we take the average of $T^*_\Phi(1)$ and $T^*_\Phi(2)$ and consider it as an upper bound to the balancing time while minimizing the energy dissipation using Eq. (*OPT-DL-ED*). Then, we consider the obtained minima as an upper bound to the energy dissipation and minimize the balancing time using Eq. (*OPT-EC-BT*) to obtain a Pareto-optimal solution. The values of the energy dissipation and the balancing time, thus obtained, are denoted as $E_D(4)$ and $T^*_\Phi(4)$ respectively.

We also study a state-of-the-art balancing algorithm, i.e., *HRT*, where balancing is performed in fixed time steps, and in each step, a set of charge transfer pairs are iteratively determined based on a heuristic. In each iteration, *HRT* [256] identifies the cell with the maximum charge level among the set of available cells. If the identified cell can feasibly transfer charge to either side, then it selects the side with the lower average charge as the direction of charge transfer. If only one direction is feasible then it is automatically selected. However, if no direction is feasible then the cell is disregarded for subsequent iterations. Now, among the feasible partner cells in the direction of charge transfer, one with the minimum charge level is selected for charge transfer. Note that when two cells are paired for charge transfer then the cells in between them become unavailable to be scheduled in the same time step.

For the two balancing architectures and the five balancing algorithms, there are ten balancing options as follows: (i) *NO-MIN-ED*, (ii) *NO-MIN-BT*, (iii) *NO-HALF-ED*, (iv) *NO-HALF-BT*, (v) *NO-HRT*, (vi) *NN-MIN-ED*, (vii) *NN-MIN-BT*, (viii) *NN-HALF-ED*, (iX) *NN-HALF-BT*, and (x) *NN-HRT*.

Linear programming (LP) relaxation: In Section 6.3.1 and Section 6.4.2 respectively, Eq. (*OPT-ED*), Eq. (*OPT-BT*), Eq. (*OPT-EC-BT*), and Eq. (*OPT-DL-ED*) are formulated as MILP models because the variables $c_{i,j}$s can only take integer values while the variables $Q_{i,f}$s can also take decimal values. However, we relax the integrality constraint on $c_{i,j}$s to solve them as LP problems, as it reduces the time complexity from NP-hard to polynomial time. We round up the values of $c_{i,j}$s, obtained from solving the LP problems, to the next integer. For the setup under study with $\widehat{d} = 6$ and $N_B = 96$, the number of feasible charge transfer pairs are 1288. This implies that by rounding up we can increase at most 1288 charge transfer cycles. However, charge transmitted (or received) per cycle Q_{tx} (or Q_{rx}), energy dissipated per cycle E_Δ and the duration of a charge transfer cycle T^C are in the range of $0.1\,\mathrm{mC}$ to $1\,\mathrm{mC}$, $0.01\,\mu\mathrm{Wh}$ to $0.1\,\mu\mathrm{Wh}$, and $10\,\mu\mathrm{s}$ to $100\,\mu\mathrm{s}$, respectively. Moreover, the total number of charge transfer cycles required for charge equalization is in the range of hundreds of millions. Thus, the impact of these 1288 additional cycles on the final charge levels $Q_{i,f}$s, the energy dissipation E_D, and the balancing time T^*_Φ, is negligible.

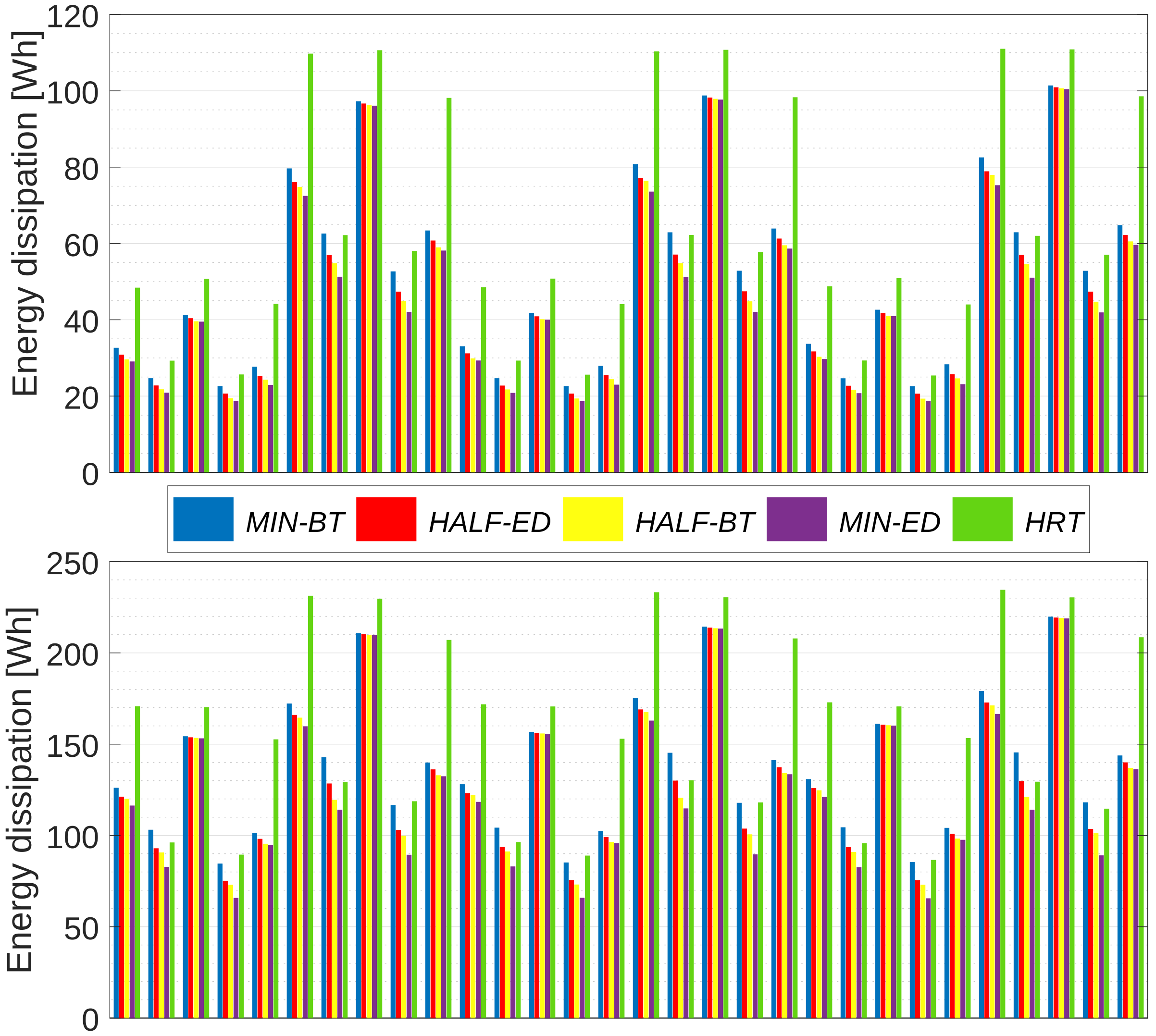


Figure 6.5: Energy dissipation in 60 random balancing scenarios (30 in the top plot and 30 in the bottom plot) using different scheduling algorithms for neighbor-only balancing architecture. The energy dissipation obtained using *MIN-ED* is the minimum in all the cases.

Performance evaluation of *MIN-ED*: We apply *MIN-ED* to calculate the minimum possible energy dissipation $E_D(1)$ during charge equalization for the 60 random cases while considering both *NO* and *NN* architectures. Note that *MIN-ED* guarantees optimality on energy dissipation and therefore, obviously results in a lower dissipation than all other scheduling algorithms for both *NO* and *NN* architectures in all scenarios as can be seen in Figure 6.5 and Figure 6.6 respectively. We further compare *MIN-ED* with *HRT* in Table 6.1. For the *NO* architecture, *MIN-ED* can reduce the energy dissipation on average by $25.92\,\%$ or $27.24\,\mathrm{Wh}$, while the maximum reduction obtained is $48.17\,\%$ or $74.62\,\mathrm{Wh}$. On the other hand, the average reduction

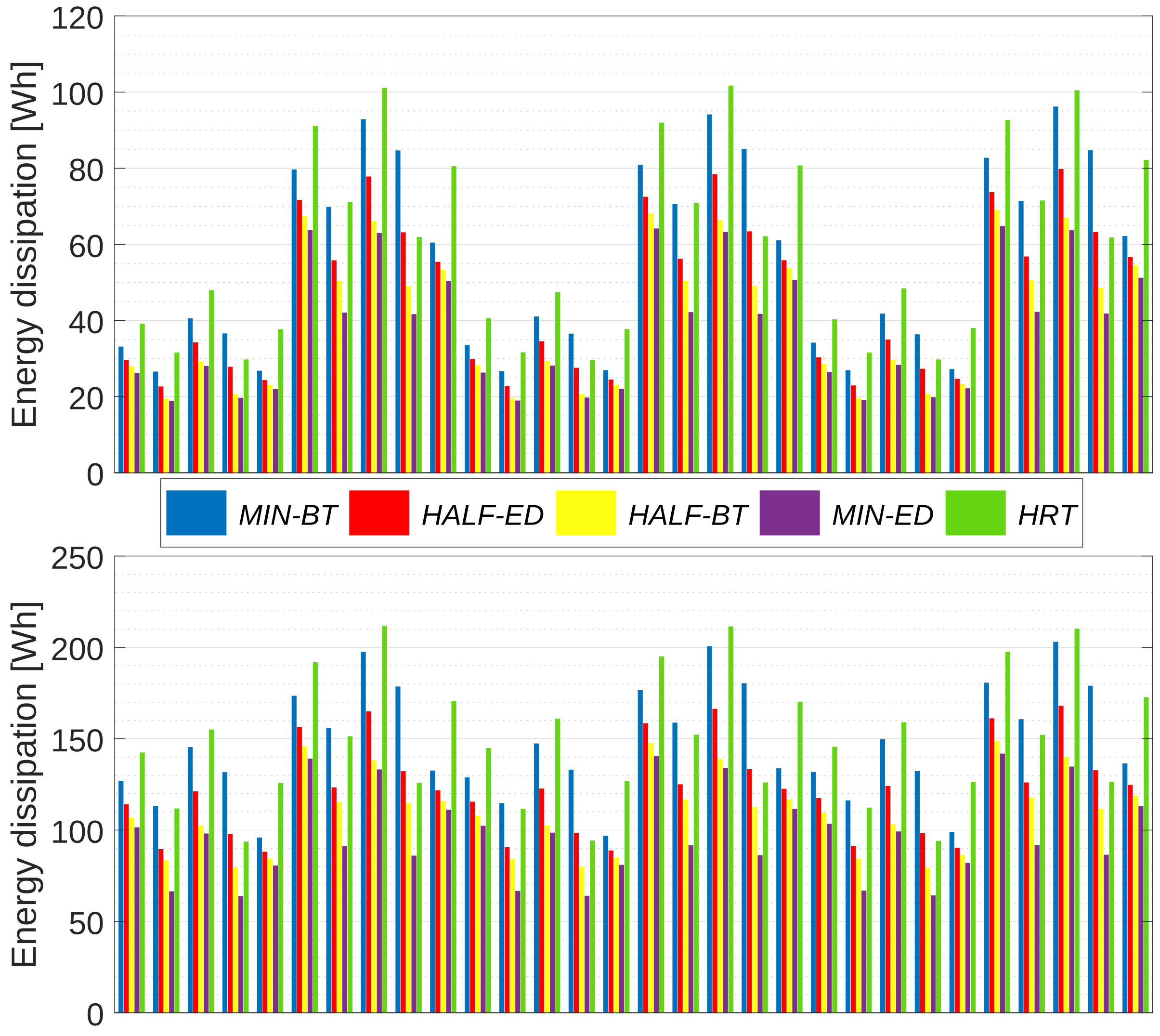


Figure 6.6: Energy dissipation in 60 random balancing scenarios (30 in the top plot and 30 in the bottom plot) using different scheduling algorithms for non-neighbor balancing architecture. The energy dissipation obtained using *MIN-ED* is the minimum in all the cases.

Table 6.1: Comparing *MIN-ED* and *HRT* in terms of energy dissipation and balancing time.

		NO-MIN-ED vs *NO-HRT*			*NN-MIN-ED* vs *NN-HRT*		
		Min	Avg	Max	Min	Avg	Max
Reduction in Energy Dissipation	in Wh	6.75	27.24	74.62	9.9	36.49	78.6
	in %	5	25.92	48.17	27.53	35.75	41.8
Reduction in Balancing Time	in h	0.01	1.75	4.34	-1.79	-0.39	0.22
	in %	0.48	29.69	43.02	-28.06	-11.12	5.34

obtained using *MIN-ED* on the *NN* architecture is $35.75\,\%$ or $36.49\,\mathrm{Wh}$ and the maximum obtained reduction is $41.8\,\%$ or $78.6\,\mathrm{Wh}$.

By applying *MIN-ED*, we also get a balancing time $T^*_\Phi(1)$. Note than $T^*_\Phi(1)$ is the maximum among all the balancing times obtained by applying Pareto-optimal balancing options, i.e., *MIN-ED*, *MIN-BT*, *HALF-ED*, and *HALF-BT* respectively. This can be also seen in Figure 6.7 and Figure 6.8 respectively. Furthermore, it can be observed in Table 6.1 that for the *NN* architecture, the balancing time obtained using *MIN-ED* is larger than that obtained using *HRT* on average by $11.12\,\%$ or $0.39\,\mathrm{h}$. This shows that towards minimizing the energy dissipation, charge transfers over a long distance are selected by *MIN-ED* as they are more energy-efficient compared to shuttling charge over multiple intermediate transfers. However, such transfers significantly limit concurrent transfers, therby increasing the overall balancing time. On the other hand, for the *NO* architecture, *MIN-ED* always performs better than *HRT* in terms of balancing time by $29.69\,\%$ or $1.75\,\mathrm{h}$ on average. This is because the energy-efficient charge transfer pairs selected by *MIN-ED* for the *NO* architecture do not increase the balancing time as they are only between neighbors and cannot significantly influence the parallel execution of other charge transfers.

Table 6.2: Comparing *MIN-BT* and *HRT* in terms of energy dissipation and balancing time.

		NO-MIN-BT vs *NO-HRT*			*NN-MIN-BT* vs *NN-HRT*		
		Min	Avg	Max	Min	Avg	Max
Reduction in Balancing Time	in h	0.62	2.66	6.22	0.09	1.2	3.22
	in %	36.24	46.97	58.99	12.82	36.48	51.12
Reduction in Energy Dissipation	in Wh	-16.04	17.67	67.14	-54.3	2.86	37.9
	in %	-12.41	15.27	37.34	-43.13	2.23	28.98

Performance evaluation of *MIN-BT*: Towards minimizing the balancing time, we apply *MIN-BT* on the 60 random cases for both *NO* and *NN* architectures respectively. *MIN-BT* guarantees to determine the global minima on the balancing time and therefore, obviously achieves the lowest balancing time among all balancing options as can be seen in Figure 6.7 and Figure 6.8 respectively. We compare the performance of *MIN-BT* with that of *HRT* in terms of balancing time and energy dissipation respectively and tabulate the results in Table 6.2. For the *NO* architecture, *MIN-BT* improves the balancing time by $46.97\,\%$ or $2.66\,\mathrm{h}$ on average as compared to *HRT* while for certain cases, it can even reduce the balancing time upto $6.22\,\mathrm{h}$, i.e., from $11.04\,\mathrm{h}$ to $4.82\,\mathrm{h}$. Similarly, for the *NN* architecture, the average reduction in balancing time as achieved by *MIN-BT* is $36.48\,\%$ or $1.2\,\mathrm{h}$ and the maximum reduction is $51.12\,\%$ or $3.22\,\mathrm{h}$.

Using *MIN-BT*, we get a value for energy dissipation $E_D(2)$ that is also higher than those obtained by other Pareto-optimal algorithms, i.e., *MIN-ED*, *HALF-ED*, and *HALF-BT*. This is also evident in Figure 6.5 and Figure 6.6 respectively. As given in Table 6.2, *MIN-BT* performs better than *HRT* on average also in terms of energy dissipation for both *NO* (by $15.27\,\%$ or $17.67\,\mathrm{Wh}$) and *NN* (by $2.23\,\%$ or$2.86\,\mathrm{Wh}$) architectures. However, note that for certain cases, energy dissipation obtained using *HRT* is lower than that obtained from *MIN-BT*. This is because *MIN-BT* might select time-efficient charge transfers exploiting the capabilities of

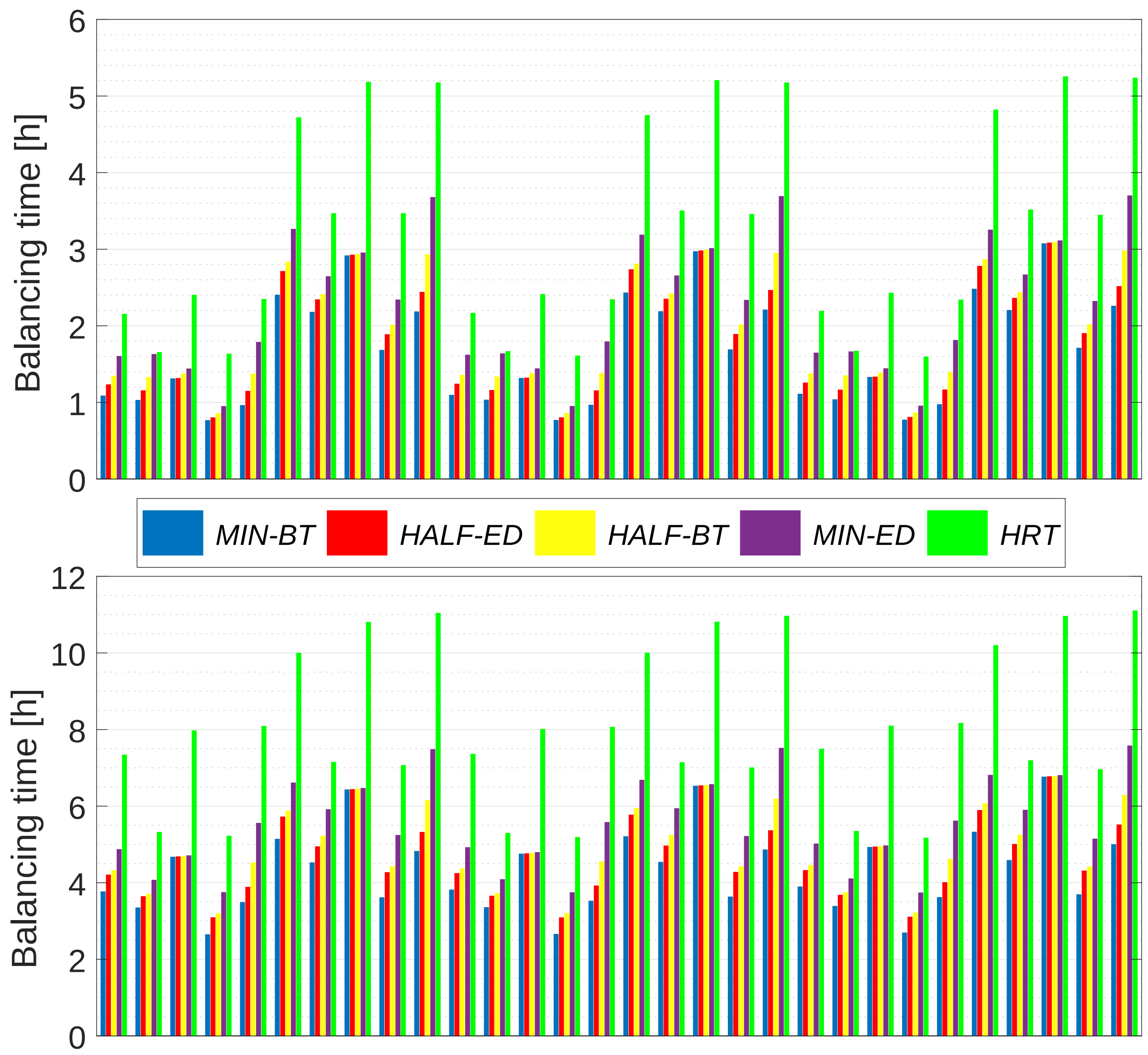


Figure 6.7: Balancing time in 60 random balancing scenarios (30 in the top plot and 30 in the bottom plot) using different scheduling algorithms for neighbor-only balancing architecture. The balancing time obtained using *MIN-BT* is the minimum in all the cases.

the balancing architectures to support concurrent transfers. However, such transfers might not always be energy-efficient.

Pareto-Optimal charge transfer schedules: For our experiments, we have considered four Pareto-optimal balancing options, i.e., *MIN-ED*, *MIN-BT*, *HALF-ED*, and *HALF-BT* respectively, as defined earlier in this section. *MIN-ED* gives the minimum energy dissipation $E_D(1)$ while *MIN-BT* schedules charge transfers in minimum time $T^*_\Phi(2)$. In Table 6.3, we compare the energy dissipation and balancing time of the Pareto-optimal schedules synthesized using the four balancing options with respect to the minimum values, i.e., $E_D(1)$ and $T^*_\Phi(2)$ respectively.

It can be observed in Figure 6.5 and Figure 6.6 respectively, that the energy dissipation $E_D(4)$ in case of *HALF-BT* is less than $E_D(3)$ that is obtained for *HALF-ED* in all the cases, i.e.,

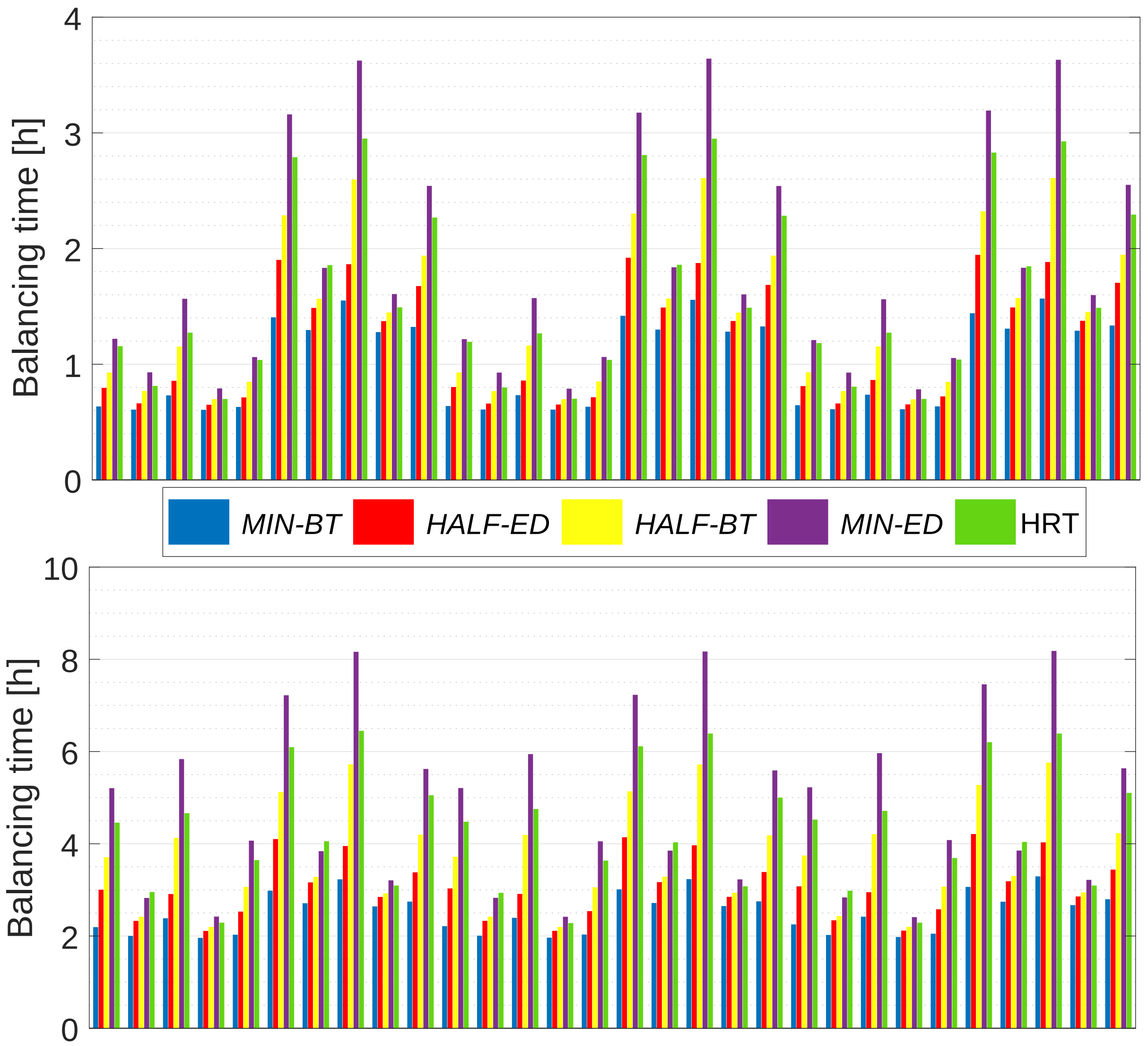


Figure 6.8: Balancing time in 60 random balancing scenarios (30 in the top plot and 30 in the bottom plot) using different scheduling algorithms for non-neighbor balancing architecture. The balancing time obtained using *MIN-BT* is the minimum in all the cases.

$E_D(4) \leq E_D(3)$. The statistics presented in Table 6.3 also suggests likewise where (i) $E_D(4)$ is on average $4.08\,\%$ and $10.78\,\%$ more than the minimum $E_D(1)$ for *NO* and *NN* architectures respectively, and (ii) $E_D(3)$ is $7.24\,\%$ and $25.73\,\%$ more than the minimum on average in case of *NO* and *NN* architectures respectively. When the four balancing options are used for the 60 random balancing scenarios, we can write $E_D(1) \leq E_D(4) \leq E_D(3) \leq E_D(2)$. Furthermore, we can note from $E_D(2)$ obtained using *MIN-BT*, that the energy dissipation can be varied up to $31.36\,\mathrm{Wh}$ for the *NO* architecture and $94.08\,\mathrm{Wh}$ for the *NN* architecture. For the *NO* architecture, the variation is not very high as the choices for charge transfers are limited to only neighbors. For any cell, the algorithm can choose between two neighbors while the energy dissipation values per cycle with either of the neighbors are almost equal. For the *NN* architecture,

Table 6.3: Comparing Pareto-optimal scheduling algorithms.

			NO architecture			*NN* architecture		
			Min	Avg	Max	Min	Avg	Max
MIN-ED	$T^*_\Phi(1) - T^*_\Phi(2)$	in h	0.04	0.89	2.66	0.17	1.59	4.94
		in %	5.79	34.51	86.41	20.54	83.76	154.14
MIN-BT	$E_D(2) - E_D(1)$	in Wh	0.97	9.57	31.36	4.82	33.64	94.08
		in %	0.44	14.48	32.69	19.08	52.05	109.24
HALF-ED	$E_D(3) - E_D(1)$	in Wh	0.48	4.79	15.68	2.37	16.68	47.04
		in %	0.22	7.24	16.24	9.4	25.73	54.62
	$T^*_\Phi(3) - T^*_\Phi(2)$	in h	0.01	0.26	0.66	0.04	0.37	1.15
		in %	0.14	9.46	19.79	6.71	19.56	37.62
HALF-BT	$E_D(4) - E_D(1)$	in Wh	0.11	2.82	12.12	0.59	7.36	28.82
		in %	0.09	4.08	13.63	3.1	10.78	33.57
	$T^*_\Phi(4) - T^*_\Phi(2)$	in h	0.02	0.44	1.33	0.09	0.8	2.49
		in %	0.3	17.26	43.16	10.35	42.25	77.38

there is a significant trade-off possible with energy dissipation considering that the maximum energy dissipation is around $200\,\mathrm{Wh}$.

Figure 6.7 and Figure 6.8 show that $T^*_\Phi(2) \le T^*_\phi(3) \le T^*_\phi(4) \le T^*_\Phi(1)$. Using *HALF-ED*, the balancing time is $9.46\,\%$ and $19.56\,\%$ higher than the minimum on average for the *NO* and the *NN* architectures respectively. Similarly, *HALF-BT* increases the balancing time by $17.26\,\%$ and $42.25\,\%$ respectively on average for the *NO* and the *NN* architectures. This reduces the energy dissipation appreciably to within $4.08\,\%$ and $10.78\,\%$ of the minimum respectively on average. Comparing *MIN-ED* and *MIN-BT*, we get the maximum variation in balancing time up to $2.66\,\mathrm{h}$ and $4.94\,\mathrm{h}$ for the *NO* and the *NN* architectures respectively. Again, for the *NN* architecture, the trade-off possibilities are significant considering that the maximum balancing time is around $8\,\mathrm{h}$.

Note that it is also possible to obtain other Pareto-optimal solutions by considering an energy constraint like in *HALF-ED* or by adding a time deadline similar to *HALF-BT*. Moreover, it is also possible to formulate an optimization objective as a linear combination of the balancing time and the energy dissipation and then solve the problem with different choices of weights for the objectives. Here, for each combination of weights, we obtain a Pareto-optimal solution.

Comparing neighbor-only and non-neighbor architectures: Using the proposed optimization framework, we can also compare the performance of the balancing architectures under study. By applying *MIN-ED* and *MIN-BT* respectively, we can obtain the minimum energy dissipation and the minimum balancing time for different balancing scenarios in a balancing architecture. For the *NO* and the *NN* architectures, we compare the results in Table 6.4 and in Figure 6.9 respectively. Using the *NN* architecture, the minimum energy dissipation obtained is lower than that obtained using the *NO* architecture in $95\,\%$ of the cases and on average it is

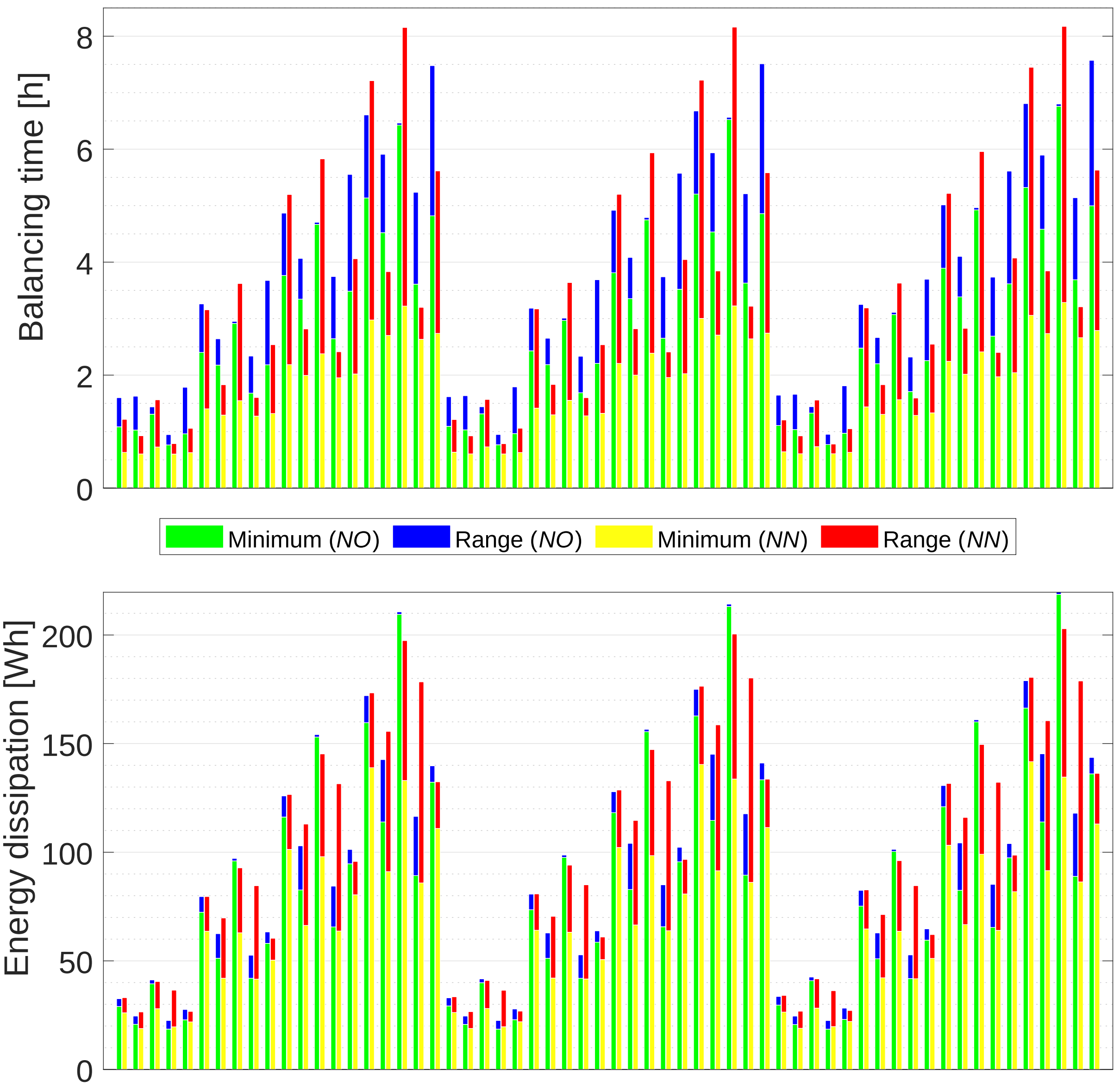


Figure 6.9: Neighbor-only balancing architecture vs non-neighbor balancing architecture. The green and the yellow bar shows the minimum possible balancing time (top plot)/energy dissipation (bottom plot) using the *NO* and the *NN* architectures respectively. The blue and the red bar shows the difference between the maximum and the minimum value of balancing time/energy dissipation using the *NO* and the *NN* architectures respectively.

$15.36\,\%$ or $16.78\,\mathrm{Wh}$ lower. Note that in the *NO* architecture, the parasitic resistances during the discharging phase is much smaller compared to that in the *NN* architecture when transferring charge between neighbors. Thus, when most of the energy-efficient pairs of cells are neighbors then it might be efficient using the *NO* architecture. With regard to the minimum balancing time obtained using *MIN-BT*, the *NN* architecture is always more efficient than the *NO* architecture and on average it allows $39.2\,\%$ or $1.23\,\mathrm{h}$ savings in time as compared to the *NO*

Table 6.4: Comparing neighbor-only and non-neighbor balancing architectures.

		NN-MIN-ED vs *NO-MIN-ED*			*NN-MIN-BT* vs *NO-MIN-BT*		
		Min	Avg	Max	Min	Avg	Max
Reduction in Energy Dissipation	in Wh	-1.18	16.78	84.1	-62.5	-7.29	16.79
	in %	-6.38	15.36	38.45	-61.9	-11.82	7.64
Reduction in Balancing Time	in h	-1.69	0.53	2.09	0.16	1.23	3.47
	in %	-26.18	16.29	44.31	21.04	39.2	51.39

architecture. Moreover, for certain cases, it can reduce the balancing time from $6.76\,\mathrm{h}$ (using the *NO* architecture) to $3.29\,\mathrm{h}$.

Furthermore, note that using *MIN-ED*, the *NN* architecture is also efficient in terms of balancing time than the *NO* architecture by $16.29\,\%$ or $0.53\,\mathrm{h}$ on average. However, in certain cases where the energy-efficient charge transfers for the *NN* architecture are more distant, balancing using the *NO* architecture might be quicker due to more number of concurrent charge transfers. On the other hand, the *NN* architecture can improve the balancing time up to $44.31\,\%$ or $2.09\,\mathrm{h}$ as compared to the *NO* architecture, when *MIN-ED* is used.

MIN-BT always offers a lower balancing time using the *NN* architecture as compared to the *NO* architecture, however, this comes at the cost of a higher energy dissipation up to $61.9\,\%$ or $62.5\,\mathrm{Wh}$. Note that towards minimizing the balancing time, *MIN-BT* might select pairs of cells that are close to each other such that the number of parallel transfers can be increased. However, transferring charge over a shorter distance is more energy-efficient using the *NO* architecture than the *NN* architecture.

In Figure 6.9, we show the minimum and the maximum possible Pareto-optimal values of balancing time (top plot) and energy dissipation (bottom plot) respectively for both *NO* and *NN* architectures. It can be observed in the figure that, in most cases, using the *NN* architecture we can obtain a shorter balancing time than the *NO* architecture while doing a Pareto optimization. However, to obtain a lower energy dissipation using the *NN* architecture, one must not optimize too much for balancing time as this would force the optimization framework to select pairs of cells closer to each other, and this is not energy-efficient in the *NN* architecture. In general, it can be concluded that the *NN* architecture is more efficient than the *NO* architecture both in time and energy if appropriately optimized. However, it must be also noted that the *NN* architecture is more expensive than the *NO* architecture due to more number of MOSFET switches, and therefore, in the future, it will be interesting to perform a utility analysis for the *NN* architecture.

Different balancing ranges, peak currents and balancing thresholds: We evaluate the impact of the balancing range ($\widehat{d}$), the peak current (I_{peak}), and the balancing threshold Q_{th}, on balancing time and energy dissipation. We obtain the minimum energy dissipation using *MIN-ED* and we apply *MIN-BT* to determine the minimum balancing time. We evaluate 20 randomly generated initial charge distributions with mean $mn = 50\%$ and standard deviation $sd = 2\%$. We vary (i) $\widehat{d}$ from 0 to 6, (ii) I_{peak} from $1\,\mathrm{A}$ to $12\,\mathrm{A}$ (at an interval of $1\,\mathrm{A}$), and (iii) Q_{th} from $100\,\mathrm{C}$ to $1000\,\mathrm{C}$ (at an interval of $100\,\mathrm{C}$). The mean energy dissipation and the mean balancing time for these variations are plotted in Figure 6.10, Figure 6.11, and Figure 6.12 respectively.

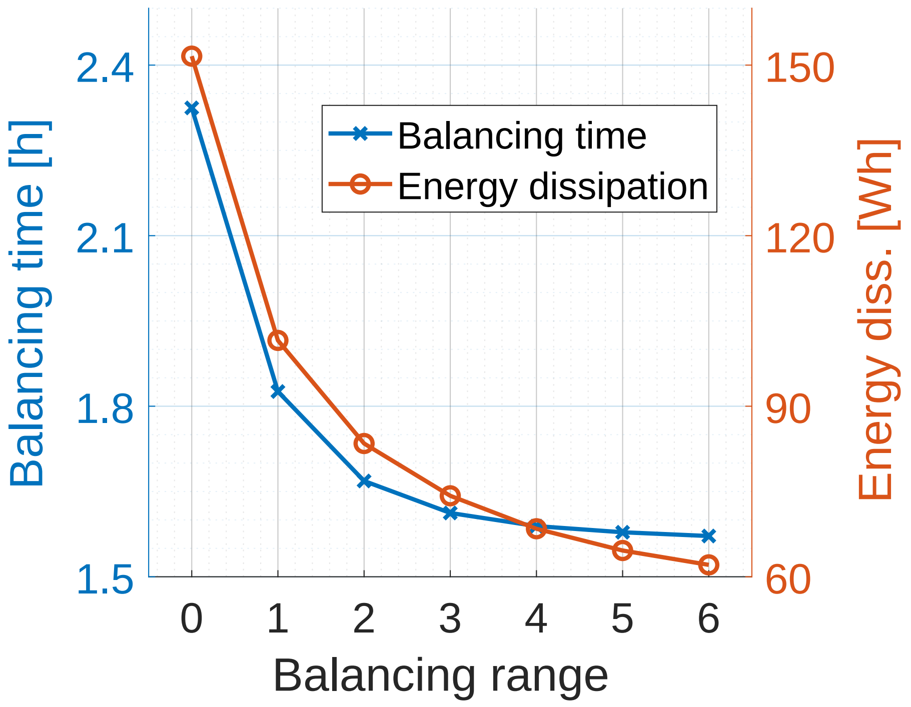


Figure 6.10: Variation in energy dissipation and balancing time with balancing range.

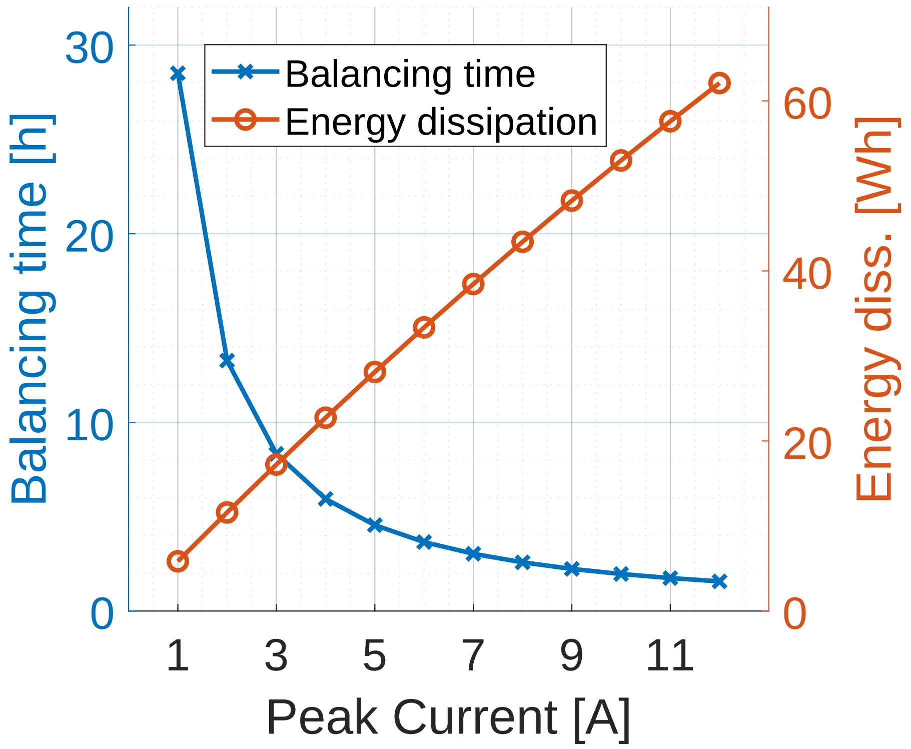


Figure 6.11: Variation in energy dissipation and balancing time with peak current.

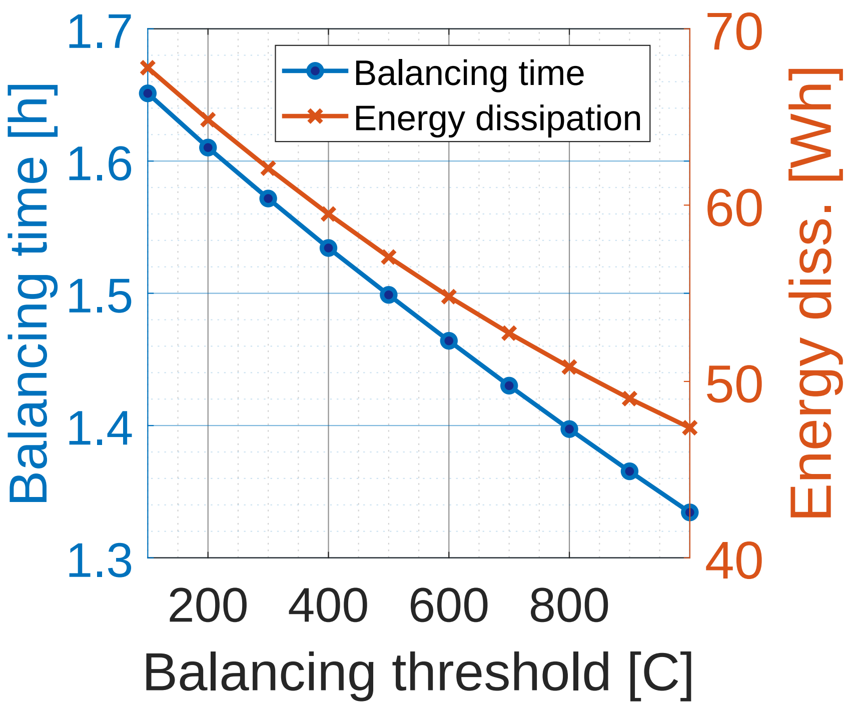


Figure 6.12: Variation in energy dissipation and balancing time with balancing threshold.

It can be observed that with increase in $\widehat{d}$, the minimum balancing time reduces, however, the reduction is not appreciable for $\widehat{d} > 3$. Similarly, the minimum energy dissipation also decreases exponentially with $\widehat{d}$. With increase in I_{peak}, balancing time decreases exponentially while the energy dissipation increases linearly. Thus, when there is a longer time available for charge equalization, e.g., when the car is in the garage overnight, it might be possible to use lower balancing current to reduce energy dissipation. Moreover, at higher values, the improvement in balancing time with increase in I_{peak}, is negligible while the increase in energy dissipation is still significant. Note that to make provision for higher balancing range and peak current, cost pertaining to circuit components increase significantly, however, they do not offer any appreciable benefit in terms of energy dissipation or balancing time after a certain point.

With increase in Q_{th}, both energy dissipation and balancing time decrease linearly. This is expected because less balancing effort is required as we allow a higher variation in the final charge levels of the cells in the battery pack. Note that our proposed framework can also be used to find the maximum balancing threshold that can be reached for a given deadline on the balancing time.

Active vs passive cell balancing As shown in Figure 6.13, the energy dissipation for passive balancing is at least 10 times more than that of active cell balancing for the 60 random balancing scenarios that we consider. The average energy dissipation in case of passive balancing is $1360\,\mathrm{Wh}$ compared to 67 Wh for active balancing. Thus, with active balancing, we save almost 6% of the battery capacity which is equivalent to approximately 5 miles of driving for a BMW i3 with $22.6\,\mathrm{kWh}$ battery [63].

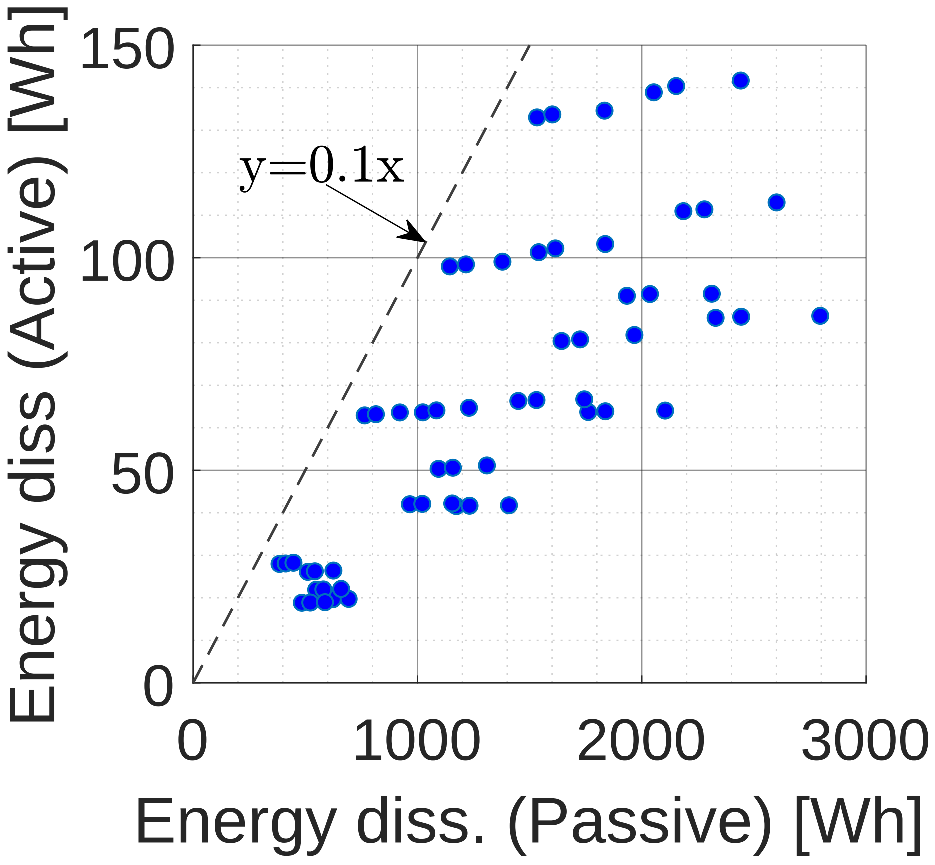


Figure 6.13: Active vs passive cell balancing in terms of energy dissipation.

Runtime of the proposed optimization: We used a computer with Intel(R) Core(TM) i7 $-$ 5600U CPU @2.60 GHz processor and 8 GB RAM to run the proposed two-stage optimization in MATLAB. We use CPLEX as the optimization solver to solve the MILP problems as well the LP (relaxed) problems. We measured the time taken by *MIN-BT* when applied for *NN* architecture on 60 randomly generated balancing scenarios. For a time limit of 300 seconds, MILP could be solved for 53 problem instances out of 60 and the average time taken to solve these 53 problems is $34.3\,\mathrm{s}$. Using LP relaxation, all the 60 problem instances could be solved in less than $0.7\,\mathrm{s}$ ($0.39\,\mathrm{s}$ on average). This time is negligible compared to a balancing time of tens of minutes to a few hours.

6.7 Related Works

Charge transfer circuit architectures: Several circuit architectures have been proposed in the power electronics domain for exchanging charge between the cells in a battery pack, where the primary goal has been to provide the maximum flexibility for charge transfer. For instance, the balancing circuit proposed in [165] supports different types of charge transfer such as cell-to-cell, a cell to a group of series-connected cells or vice-versa and from group of cells to another group of cells. A comprehensive overview of different active cell balancing architectures is provided in [261, 262]. Depending upon the energy storage element used for charge transfers, the active cell balancing architectures are classified into capacitor-based [253, 254], inductor-based [165, 256], and transformer-based [85, 257] approaches. Furthermore, the active cell balancing architectures are classified based on the offered charge transfer flexibility as neighbor-only [164] (as shown in Figure 2.6(a)) and non-neighbor charge transfer architectures [85] (as shown in Figure 2.6(c)). While the circuit architectures aim to achieve a high energy efficiency during a charge transfer by minimizing the parasitic resistances and capacitances of the circuit components, the system-level goals such as minimizing the energy dissipation and the balancing time are not considered. Moreover, the system-level performance of any active cell balancing architecture significantly depends on the control algorithm that selects the charge transfer pairs of cells.

Control algorithms for cell balancing: In addition to the circuit architectures facilitating charge transfer between the cells in a battery pack, algorithms for controlling these architectures have been proposed in the literature. The primary goal of these balancing algorithms has been to determine the *source* and *destination* cells for charge transfers such that the charge levels of all the cells in a battery pack are equalized eventually. Towards this, early works [258,259] have adopted a control-theoretic approach that guarantees asymptotic stability (i.e., charge equalization). On the other hand, other existing algorithms are based on heuristics. For instance, [256] (as explained in Section 6.6) proposes a balancing heuristic that selects pairs of cells such that the variance in the charge levels of the cells reduces with time. These aforementioned algorithms do not guarantee minimum energy dissipation or minimum balancing time. They are designed for centralized battery management topologies, where a single master controller knows the charge levels of all the cells in the battery pack and decides the source and destination cells for charge transfers. Recently, decentralized battery management approaches, where each cell is controlled by a dedicated control unit that interacts with other cell-level controllers without a central master controller, are also being studied [263]. Subsequently, decentralized active cell balancing algorithms have been proposed for both neighbor-only balancing architectures in [84] and for non-neighbor balancing architectures in [82]. These decentralized balancing algorithms are also heuristics following a similar approach to [256]. Furthermore, similar balancing heuristics [165, 255] are also proposed for controlling complex balancing architectures that can support different charge transfer schemes (such as from a single cell to a group of cells and vice versa, and also between two groups of cells). Although several balancing algorithms for controlling different balancing architectures considering different topologies for battery management systems have been proposed in the literature so far, however, none of them guarantee optimality in terms of energy dissipation or balancing time. Thus, the full potential of the existing balancing architectures is not known. Moreover, these algorithms are strictly tied to the capabilities of the underlying balancing architecture, i.e., the algorithms proposed for the neighbor-only architecture are not directly scalable for the non-neighbor architecture. On the other hand, we have shown that our proposed scheduling framework for cell balancing can be used for both neighbor and non-neighbor architectures, while extending this framework for other complex architectures can be considered in the future. For our framework, we assumed a centralized battery management system, however, note that our framework can also be used as a benchmark for evaluating the performance of different balancing heuristics proposed for decentralized systems. Moreover, for future battery management systems, this schedule computation can also be performed on the cloud and the charge transfer schedules for a cell can be communicated to the corresponding cell-level controller. Finally, our framework also provides guidelines for designing the balancing architecture, e.g., suggesting an appropriate balancing range ($\widehat{d}$) that the architecture should support.

Real-time energy systems: Real-time systems community has studied energy systems at multiple abstraction-levels. On individual cell-level, several reconfiguration architectures and their corresponding scheduling strategies have been studied in [264, 265]. These approaches improve the runtime of a battery powered application, by dynamically modifying the electrical connection of individual cells in a battery pack depending upon the real-time changes in the load. On one level higher, batteries are considered as a single energy storage unit and several real-time scheduling algorithms for their charging, discharging and rest periods have been proposed [266–268]. Improvements in the system software design [269] and controller design [81] based on the battery capacity have been studied. Further in the higher abstraction-level, event-driven scheduling strategies have been proposed in [270] to optimize the design of a large-scale EV charging stations.

In summary, while several balancing algorithms exist in the literature, they do not guarantee optimality in terms of energy dissipation and balancing time. While the real-time systems community have proposed multiple scheduling techniques for charging or discharging the battery pack, the problem of improving the usable capacity of a battery pack by performing charge redistribution has not been studied till now. In this thesis, we propose a two-stage optimal scheduling framework that determines the charge transfer schedule for charge equalization in a battery pack while minimizing the energy dissipation and the balancing time respectively.

6.8 Conclusion

In this work, for the first time, we formulate the active cell balancing problem from a real-time systems perspective. We propose a multi-stage scheduling framework to minimize the energy dissipation and the balancing time respectively for charge equalization in a battery pack. In the first stage, we formulate an MILP problem to determine the optimal set of charge transfer pairs and the number of cycles for which each pair must perform charge transfer. In the second stage, we apply the minimum vertex coloring algorithm for chordal graphs to synthesize a schedule for the charge transfers obtained from Stage 1.

Our experimental results show that there exists a trade-off between the balancing time and the energy dissipation. Moreover, we also show that other control knobs like peak balancing current can be tuned based on requirements. In the future, a more comprehensive framework can be developed that can optimally tune different control knobs for each balancing scenario. Note that our proposed framework assumes a centralized BMS. The optimal results obtained for different balancing scenarios using our framework can be studied to determine if there is any pattern that can be exploited to select charge transfer pairs in case of a decentralized BMS. Furthermore, these results can also be studied to optimize the design of charge transfer circuit architectures with respect to cost and performance.

7 Concluding Remarks

7.1 Lessons Learnt

This thesis primarily advocates comprehensive modeling of CPSs towards more optimal designs. Such approaches are particularly useful in safety-critical and resource-constrained CPSs, where improving control performance and reducing resource usage are important. However, such a design paradigm is faced with several challenges such as the heterogeneity of the system models, the large number of design dimensions, conflicting design objectives, and the disjoint set of available design tools. Through our works, we show that these challenges can be effectively tackled by (i) systematically analyzing the models, (ii) deriving interfaces between the models, (iii) exploiting the interfaces to partition the holistic design problem into several partitions, and (iv) solving each sub-problem using the most suitable mathematical tool. Note that in our works, we do not try to model the whole system using a unified language unlike in the theory of hybrid systems that is based on the unification of differential equations and automata [271]. In this section, we briefly revisit our works and give the intuitions behind the design approaches. We believe that these approaches can be extended to other CPS settings.

Multi-objective co-optimization for distributed CPSs: We have considered distributed time-triggered implementation of controllers. For such a setting, we design controllers and their platform implementation in a holistic framework while co-optimizing the average control performance and the resource usage. As a communication bus, we take the example of FlexRay. We assume that the messages are transmitted on TDMA slots in the FlexRay static segment. We know that FlexRay messages can be sent using only a limited number of repetition rates. We exploit this fact to constrain the choices of sampling period corresponding to the permissible set of repetition rates. Controller design depends on the sampling period and the closed-loop delay, while it is oblivious to the exact schedule of the tasks and messages in the implementation. For a given sampling period, we fix the value of delay as equal to one sampling period to allow maximum flexibility in the implementation. Thus, we have limited number of combinations of sampling period and delay. For each combination, we can predesign the optimal controller. Thus, we have a set of prospective controllers using which an application can be implemented. Note that the controller design problem that we have considered in this thesis, does not fit into any standard closed-form optimization framework, and therefore, we use evolutionary method, i.e., PSO.

We further formulate a multi-objective optimization problem considering control requirements and implementation-specific and application-level constraints. As the two objectives are conflicting, we aim to generate a Pareto front. Exploiting the discrete nature of resource usage, we split the co-design problem into several sub-problems with the average control performance as the only objective and an equality constraint on the resource usage, thereby, reducing the problem complexity. We note that both control performance and resource usage will only depend on the choices of sampling periods. Therefore, we can further reduce the problem complexity by partitioning each single-objective optimization problem into two nested layers. In the outer layer, we solve an MILP to find a set of sampling periods that maximize the control performance while matching the discrete value of the resource usage. In the inner layer, we find a feasible set of task and message schedules by solving an ILP according to the obtained sampling periods from the outer layer. Our approach has shown to easily scale to a reasonable-sized system comprising 24 applications mapped on to 12 ECUs. In summary, this work shows that it is possible to exploit problem characteristics and control-theoretic knowledge to partition a complex optimization problem into several smaller sub-problems without compromising the design optimality.

Tool integration for automated synthesis and implementation of distributed CPSs: We have further developed an integrated toolchain for automated synthesis of distributed control software that uses the proposed control-platform co-design approach. In this work, the main goal has been to minimize manual interventions in the process of software development. Here, we study the conventional approach for software development in the automotive industry, in particular, using the Elektrobit tools SIMTOOLS/SIMTARGET for platform modeling and MATLAB/Simulink for controller design and modeling. We exploit the advantages of the COTS tools, e.g., platform modeling and code generation, while offering tools to automate other important steps such as modeling the software implementation of a controller, specification extraction, control-platform co-design, and parameter writeback. Using Simulink and SIMTOOLS blocks, we create a library offering template blocks that can be configured to automatically generate a plant model and the software implementation of a controller respectively. These blocks are used for specifying the system while the platform architecture is specified in SIMTOOLS. Now, SIMTOOLS can export the platform specification as a FIBEX file, while a Simulink model can be parsed using standard MATLAB functions. Using these features, we can automatically extract the control/architecture specification. Based on the extracted details, we can apply the proposed control-platform co-design approach to generate a Pareto front. Now, the developer can select the Pareto point to be implemented. While the controller gains and the task schedules can be automatically written into Simulink and SIMTOOLS models respectively, we generate a CSV file for the message schedules that can be directly imported into SIMTOOLS model. Finally, the binary files can be generated automatically using SIMTARGET and Simulink RTW. The ECUs can be then flashed using the respective binary files. In summary, this work shows that different tools that are used to design different parts of a CPS can be interfaced if studied systematically.

Tighter dimensioning of multi-resource CPSs with control performance guarantees: We consider a setting where a controller can be implemented using different types of resources, i.e., high-quality and low-quality resources. Using high-quality resources, faster rejection of disturbances is possible, however, they are expensive. For such a setting, we study the resource dimensioning problem, i.e., the minimum amount of such expensive resources required to meet the requirements of all applications. A bimodal switched controller is considered that operates either in the high-cost mode using high-quality resources or in the low-cost mode using low-quality resources. For such controllers, we propose static and dynamic scheduling schemes.

First, an algorithm is developed to determine a periodic pattern in which a bimodal controller must switch, such that it meets the control requirements while using the minimum amount of high-quality resources. This algorithm analyzes the dynamics of the physical system for different switching patterns. Now, given these patterns, we can formulate an SMT problem to synthesize a static schedule for the bimodal controllers sharing the high-quality resources. The obtained static schedule also has the maximum extensibility.

We further propose a dynamic scheduling policy where a controller operates in the low-cost mode during the steady state and requests to switch to the high-cost mode only when disturbed. Dynamic priority based arbitration of requests from contesting applications is proposed. Based on the proposed scheduling scheme, we study the physical dynamics to derive switching constraints from the control requirements. Based on these constraints, we tightly dimension the high-quality resources using a nested two-layer approach. In the outer layer, a first fit heuristic is used to determine a many-to-one mapping of applications to resources. Here, a mapping of an application to a resource means that whenever the application is disturbed, it can only use the allocated resource. Now, for a given set of applications mapped to a single resource, we formulate a formal verification problem in the inner layer to verify if all applications will meet their respective requirements in the worst-case. Here, we model the applications and the scheduling algorithm as a network of TA and use model checking for the verification.

In summary, this work brings into attention that an interdisciplinary study of control theory, scheduling, formal verification, and advance optimization techniques might be required to effectively solve many CPS design problems.

Energy- and time-optimal active cell balancing in battery packs: In this thesis, we also study high-power battery packs, in particular, the problem of active cell balancing in such a pack, and propose algorithms to determine schedules that optimize energy dissipation and balancing time respectively. We show that the cell balancing problem can be partitioned into two subproblems. Here, we study the charge transfer model between a pair of cells using state-of-the-art charge transfer circuit architectures. We establish that the charge equalization in a battery pack only depends on the amount of time each feasible pair of cells transfers charge and not on the exact order of charge transfer. Similarly, we also derive a linear expression for the total energy dissipation during a cell balancing session that is independent of the schedule of charge transfer. This enables us to formulate an MILP problem to determine the duration of charge transfer between each feasible pair of cells such that the battery pack is balanced and the energy dissipation is minimized. We further show that an MVC problem can be formulated to schedule the obtained charge transfers in minimum time. Note that the obtained charge transfer schedule using this two-stage approach will only guarantee minimum energy dissipation. This is because we start with a set of charge transfers in the second stage, and hence, we do not explore the whole design space when minimizing the balancing time. Thus, the obtained balancing time is the minimum possible only for the given set of charge transfers.

Balancing time is directly influenced by the charge transfer schedule, hence, can we minimize the balancing time using the same framework? Towards this, we study the scheduling problem in the second stage in detail. That is, we prove that the conflict graph, to which the MVC algorithm is applied, possesses special features that can be exploited to derive a linear expression for balancing time in terms of the charge transfer times of the feasible pairs. Thus, we can directly use the derived expression in the MILP in the first stage to minimize the balancing time directly. Now, we obtain a set of charge transfers from the first stage that will guarantee minimum balancing time when scheduled using the MVC algorithm. Note that here we exploit the characteristics of the scheduling problem in the second stage to formulate the MILP problem in the first stage. In summary, this work emphasizes the need for novel hybrid optimization techniques to tackle the heterogeneous models involved in CPS designs.

7.2 Future Research Directions

While this thesis lays the foundations for comprehensive modeling and design of CPSs, there are still several aspects of CPSs design that have received little or no attention in the literature. In this section, we will list several future research directions towards more holistic design considerations for CPSs.

Nonlinear physical dynamics: While in this thesis, we have studied linear dynamics only, nonlinear plants are common in several domains of CPSs, including avionics and automotive. However, co-synthesis of controllers that handle nonlinear plants, and their platform implementations is still not considered in the literature. There have been some works in the stabilization and control of nonlinear plants using abstractions of the implementation platform. Many related works in this direction are based on concepts like input-to-state stability [272, 273], small gain theorem [274], passivity [275] and feedback linearization [276].

Fuzzy-model based analysis and control of nonlinear systems have also received significant attention. Among different fuzzy models, nonlinear systems fit well into Takagi-Sugeno (T-S) models [277]. In such a model, at each sampling time the system is represented as an averaged linear model. Based on T-S models, there have been works that consider network-specific properties like packet dropout, signal quantization and time delays. Towards considering packet drops, data loss in T-S fuzzy-based systems is modeled as a Bernouli process. Correspondingly, (i) stability is studied based on a common quadratic Lyapunov function and a fuzzy Lyapunov function [278] and (ii) design of $\mathcal{H}_\infty$ state feedback control [279], static/dynamic output feedback control [280, 281], observer-based output feedback reliable control [282] and model predictive control [283] are proposed. Towards time-delayed nonlinear systems, most works study properties like maximum allowable delay bound [284], maximum allowable transfer interval [285] and delay distribution [286, 287]. Corresponding to these parameters, the stability and control of nonlinear systems can be evaluated. Furthermore, there have been works that consider the impact of network-induced signal quantization on T-S fuzzy-based nonlinear systems. They use abstracted platform models based on time-invariant logarithmic quantizer [288] or time-varying quantizer [289] to study stabilization and control problems. However, we may point out again that all these works start with platform abstractions. Therefore, in the context of CPSs, we can leverage on these advanced theories of fuzzy-based control. We can consider co-synthesis of platform parameters and controllers by systematically deriving an interface between the platform parameters and the abstraction models.

Time-varying physical dynamics: This thesis mainly shows results for time-invariant plant dynamics. However, there are several real-world physical processes with time-varying dynamics especially in flight and vehicular controls due to the effect of weather conditions like wind speed and direction on the physical behavior. An adaptive controller can typically manipulate the control gains online based on the changing plant dynamics. Therefore, it can stabilize the system in the event of unforeseen environmental variations. However, these techniques have not been considered for safety-critical systems in the past as it is challenging to quantify the transient performance of an adaptive control loop.

A popular adaptive control technique is Model Reference Adaptive Control (MRAC). In this technique, the error between the output of the reference model and the actual output is fed back to adapt the control gains. In order to improve the transient performance, closed-loop reference models are considered of late. Here, the error is also fed back to change the reference model. Using such a controller, there have been a few works [290–293] that quantify the transient performance using $\mathcal{L}_2$-norm of error signals. An important research question here is: Can we co-adapt the control gains along with the platform implementation to better react to the varying plant dynamics? On the other hand, a few works [294–296] have considered using adaptive controllers to tackle time-varying platform behavior, e.g., network induced variable delay. However, comprehensive control-platform co-design approaches considering variations in plant dynamics and platform behavior are still largely missing in the literature.

Heterogeneous networks: Modern CPSs like automotive systems consist of several bus clusters connected via gateways. These bus clusters serve different functional domains, e.g., chassis, powertrain, body, and infotainment. Today, with increasing demand for ADASs and autonomous driving, the need for inter-domain interaction and communication has increased. Control applications are, therefore, implemented over heterogeneous networks. Designing such applications is not a straightforward extension of existing techniques. The problem is that different communication protocols have different timing models, and therefore, require different analysis framework. For example, CAN employs a fixed-priority non-preemptive scheduling scheme while FlexRay uses TDMA for static segment and FTDMA for the dynamic segment. Designing an application across CAN and FlexRay will require finding TDMA schedules for FlexRay messages and priorities for CAN messages. Moreover, inter-domain communication involves transmission of messages across communication gateways. This requires additional timing analysis and buffer characterization for gateways. Therefore, the design of applications across different network domains leads to increase in design dimensions and a more complicated timing analysis.

In this context, [153] has proposed a hybrid analysis framework where different timing analysis techniques can be composed together to determine, for example, end-to-end delay of a message. However, control applications over such heterogeneous networks are not yet considered. Co-synthesis of controllers, heterogeneous network schedules and gateway parameters will be challenging to explore.

Multi-core processors: ECUs with multiple processing cores are becoming increasingly more popular in embedded systems. The cores may share different hardware components, e.g., memory, I/O and on-chip bus. Simultaneous access to these shared resources may result in contention. Access to shared resources, if not properly managed or synchronized, may result in nondeterministic timing behavior that is difficult to analyze. There have been few works addressing this problem from both hardware [297, 298] and software [299] perspective.

For example, Tabish et al. have proposed a scratchpad-centric solution [154]. They have assumed that each core has its own scratchpad with size greater than any two tasks running on the core. The access to the main memory is according to a TDMA-based schedule via a direct memory access (DMA). The idea is that the code for the next task can be pre-fetched in one half of the scratchpad while the current task is running from the other half. In this approach, there is no resource contention. Additionally, the WCETs of the tasks are also reduced as the instructions are already in the scratchpad before execution. Consequently, control codes can be mapped on such an architecture to achieve higher control performance. However, large-sized and dedicated scratchpad for each core substantially increases the cost of the system which may not be acceptable in cost-sensitive domains. Therefore, we believe there is a possibility of using smaller dedicated scratchpad or shared scratchpad. Program analysis techniques may be used for appropriate memory partitioning and code mapping. Program analysis integrated with co-synthesis of controllers, processor and memory access schedules may result in an improved control performance and better load balancing across the cores.

Other emerging design considerations: In addition to complex physical dynamics and advanced platform architectures, the requirements of security, battery-awareness, and fault-tolerance have imposed new challenges in the design of resource-constrained CPSs. There are a few initial works done in these directions.

Towards secure CPSs, Zheng et al. have proposed a cross-layer design framework [300] that combines controller design and implementation along with security integration. It offers a trade-off analysis between degree of security, control performance and platform schedulability. More recently, Liang et al. in [301] have leveraged weakly-hard constraints for control application to add security monitoring tasks in the system.

Towards battery-aware CPSs, Chang et al. in [81] have proposed to design a DC motor speed controller taking battery characteristics into consideration. In the same vein, Vatanparvar et al. have proposed design of heating, ventilation and air conditioning (HVAC) control together with BMS [302]. This design improves battery lifetime and driving range of EVs while keeping vehicle climate within acceptable range.

Towards fault-tolerant CPSs, Goswami et al. in [303] have considered designing controller such that the control loop is stable to intermittent hardware faults. A hardware fault is characterized using intermittent bit flip model. The probability that a faulty sample is followed by at least N non-faulty samples is calculated using Monte Carlo simulations. Using this information, a controller is designed that guarantees stability even in the event of a fault.

It must be noted that research in these aforementioned directions is far from being complete. We envision a stronger emphasis in the future for a more comprehensive consideration of the aspects of security, battery-awareness, and fault-tolerance in the design of CPSs.

Bibliography

[1] E. A. Lee, “The Past, Present and Future of Cyber-Physical Systems: A Focus on Models,” *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.

[2] S. Chakraborty, M. A. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, “Automotive Cyber-Physical Systems: A Tutorial Introduction,” *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.

[3] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. -E. Arzen, “How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime,” *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[4] M. Al Khatib, A. Girard, and T. Dang, “Verification and Synthesis of Timing Contracts for Embedded Controllers,” in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2016.

[5] D. Roy, L. Zhang, W. Chang, S. Mitter, and S. Chakraborty, “Semantics-preserving cosynthesis of cyber-physical systems,” *Proceeding of the IEEE*, vol. 106, no. 1, pp. 171 – 200, 2017.

[6] L. Zhang, *Synthesizing Communication-Centric Automotive Cyber-Physical Systems*. Dissertation, Technische Universität München, München, 2018.

[7] F. Sagstetter, *Schedule Synthesis for Time-Triggered Automotive Architectures*. Dissertation, Technische Universität München, München, 2018.

[8] R. N. Charette, “This car runs on code,” *IEEE spectrum*, vol. 46, no. 3, p. 3, 2009.

[9] F. Simonot-Lion, “In car embedded electronic architectures: How to ensure their safety,” in *IFAC International Conference on Fieldbus Systems and their Applications (FET)*, 2003.

[10] H. J. Bergveld, W. S. Kruijt, and P. H. L. Notten, *Battery Management Systems – Design by Modeling*. Springer Netherlands, 2002.

[11] S. Narayanaswamy, *Design, Modeling and Implementation of Distributed Architectures for Modular Battery Packs*. Dissertation, Technische Universität München, München, 2018.

[12] RobertBosch GmbH, “E-Ray: FlexRay IP module.” User’s Manual, 2009.

[13] W. Zimmermann and R. Schmidgall, *Bussysteme in der Fahrzeugtechnik*. Springer Fachmedien Wiesbaden, 2014.

[14] D. John, "OSEK/VDX history and structure," in *IEE Seminar on OSEK/VDX Open Systems in Automotive Networks*, 1998.

[15] "Road vehicles — Open interface for embedded automotive applications — Part 1: General structure and terms, definitions and abbreviated terms," ISO 17356-1:2005, International Organization for Standardization, 2005.

[16] "Road vehicles — Open interface for embedded automotive applications — Part 3: OSEK/VDX Operating System (OS)," ISO 17356-3:2005, International Organization for Standardization, 2005.

[17] "OSEK/VDX time triggered operating system," Version 1.0, OSEK/VDX, 2001.

[18] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-Vehicle Networks: A Review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[19] BMW Group, *Der neue BMW 7er*. Springer Verlag, 2009.

[20] FlexRay Consortium, "Flexray Communications System Protocol Specification Version 2.1," 2005.

[21] FlexRay Consortium, "Flexray Communications System Protocol Specification Version 3.0.1," 2010.

[22] M. Rausch, *FlexRay: Grundlagen, Funktionsweise, Anwendung*. Hanser Verlag, 2008.

[23] "Road vehicles – Controller area network (CAN)," ISO 11898, International Organization for Standardization, 2015.

[24] "Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit," ISO 11898-2:2016, International Organization for Standardization, 2016.

[25] "Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface," ISO 11898-3:2006, International Organization for Standardization, 2006.

[26] F. Hartwich, "CAN with Flexible Data-Rate," in *International CAN Conference (iCC)*, 2012.

[27] "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," ISO 11898-1:2015, International Organization for Standardization, 2015.

[28] "Road vehicles — Controller area network (CAN) — Part 4: Time-triggered communication," ISO 11898-4:2004, International Organization for Standardization, 2006.

[29] K. Reif, *Automobilelektronik*. Springer Fachmedien Wiesbaden, 2014.

[30] "Road vehicles — Local Interconnect Network (LIN) — Part 1-8," ISO 17987, International Organization for Standardization.

[31] K. Borgeest, *Elektronik in der Fahrzeugtechnik*. Vieweg+Teubner Verlag, 2014.

[32] A. Grzemba, *MOST: The automotive multimedia network, from MOST25 to MOST150*. Franzis, 2011.

[33] Broadcom Corporation, "BroadR-Reach® Physical Layer Transceiver Specification for Automotive Applications V3.0," 2014.

[34] C. M. Kozierok, C. Correra, R. B. Boatright, and J. Quesnelle, *Automotive Ethernet The Definitive Guide*. Intrepid Control Systems, 2014.

[35] C. E. Spurgeon and J. Zimmerman), *Ethernet: The Definitive Guide: Designing and Managing Local Area Networks*. O'Reilly Media, 2000.

[36] "IEEE Standard for Local and Metropolitan Area Networks - System Considerations for Multisegment 10 Mb/S Baseband Networks (Section 13) and Twisted-Pair Medium Attachment Unit (MAU) and Baseband Medium, Type 10BASE-T (Section 14)," IEEE 802.3i-1990, IEEE Standards Association, 1990.

[37] "AIRCRAFT DATA NETWORK PART 7 AVIONICS FULL-DUPLEX SWITCHED ETHERNET NETWORK," ARINC 664 P7, Aeronautical Radio, Incorporated, 2009.

[38] "IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic," IEEE 802.3br-2016, IEEE Standards Association, 2016.

[39] "Time-Sensitive Networking Task Group." `http://www.ieee802.org/1/pages/tsn.html`. Accessed: 2020-01-01.

[40] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," IEEE 802.1AS-2010, IEEE Standards Association, 2010.

[41] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment: 9: Stream Reservation Protocol (SRP)," IEEE 802.1Qat-2010, IEEE Standards Association, 2010.

[42] "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: Forwarding and Queuing Enhancements for Time-Sensitive Streams," IEEE 802.1Qav-2009, IEEE Standards Association, 2009.

[43] L. Lo Bello, "The case for Ethernet in Automotive Communications," *ACM SIGBED Review*, vol. 8, no. 4, pp. 7–15, 2011.

[44] L. Lo Bello, “Novel trends in automotive networks: A perspective on Ethernet and the IEEE Audio Video Bridging,” in *Emerging Technology and Factory Automation (ETFA)*, 2014.

[45] “The Time-Triggered Ethernet Standard,” SAE AS6802, SAE International, 2016.

[46] T. Streichert and M. Traub, *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*. Springer-Verlag, 2012.

[47] M. Di Natale and A. L. Sangiovanni-Vincentelli, “Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools,” *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[48] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, N. Chang, S. Park, Y. Kim, P. Leteinturier, and H. Adlkofer, “Embedded systems and software challenges in electric vehicles,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[49] K. Becker, J. Frtunikj, M. Felser, L. Fiege, C. Buckl, *et al.*, “RACE RTE: A Runtime Environment for Robust Fault-Tolerant Vehicle Functions,” in *Critical Automotive Applications: Robustness & Safety (CARS)*, 2015.

[50] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll, “RACE: A Centralized Platform Computer Based Architecture for Automotive Applications,” in *Electric Vehicle Conference (IEVC)*, 2013.

[51] “AUTOSAR Classic Platform Standards.” `https://www.autosar.org/standards/classic-platform/`. Accessed: 2020-01-01.

[52] AUTOSAR, “AUTOSAR Classic Plazform Release 4.3.1,” 2017.

[53] “AUTOSAR.” `https://www.autosar.org`. Accessed: 2020-01-01.

[54] M. Staron and D. Durisic, *AUTOSAR Standard*. Automotive Software Architectures, Springer, Cham, 2017.

[55] IEC, “Electrical Energy Storage.” `https://www.iec.ch/whitepaper/pdf/iecWP-energystorage-LR-en.pdf`.

[56] T. M. Gür, “Review of Electrical Energy Storage Technologies, Materials and Systems: Challenges and Prospects for Large-Scale Storage,” *Energy & Environmental Science*, vol. 11, 2018.

[57] D. Ebbing and S. D. Gammon, “Electrochemistry,” in *General Chemistry*, ch. 19, Cengage Learning, 2016.

[58] M. Morris, “Comparison of rechargeable battery technologies,” *ASME Early Career Technical Journal*, vol. 11, pp. 148–155, 2012.

[59] D. Linden and T. B. Reddy, *Handbook Of Batteries 3rd Edition*. McGraw-Hill Handbooks, McGraw-Hill Professional, 2001.

[60] H. C. Hesse, M. Schimpe, D. Kucevic, and A. Jossen, “Lithium-Ion Battery Storage for the Grid—A Review of Stationary Battery Storage System Design Tailored for Applications in Modern Power Grids,” *Energies*, vol. 10, no. 12, pp. 1–42, 2017.

[61] “A PEEK INSIDE THE BATTERY OF A TESLA MODEL S.” `https://qnovo.com/peek-inside-the-battery-of-a-tesla-model-s/`. Accessed: 2020-01-01.

[62] “INSIDE THE BATTERY OF A NISSAN LEAF.” `https://qnovo.com/inside-the-battery-of-a-nissan-leaf/`. Accessed: 2020-01-01.

[63] “The Composition of EV Batteries: Cells? Modules? Packs? Let’s Understand Properly!.” `https://www.samsungsdi.com/column/all/detail/54344.html`. Accessed: 2020-01-01.

[64] K. Liu, K. Li, Q. Peng, and C. Zhang, “A brief review on key technologies in the battery management system of electric vehicles,” *Frontiers of Mechanical Engineering*, pp. 1–18, 2018.

[65] 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*, vol. 226, pp. 272 – 288, 2013.

[66] J. Cao, N. Schofield, and A. Emadi, “Battery balancing methods: A comprehensive review,” in *Vehicle Power and Propulsion Conference*, pp. 1–6, 2008.

[67] W. F. Bentley, “Cell balancing considerations for lithium-ion battery systems,” in *Annual Battery Conference on Applications and Advances*, 1997.

[68] 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*, vol. 247, pp. 332–338, 2014.

[69] M. M. Ur Rehman, M. Evzelman, K. Hathaway, R. Zane, G. L. Plett, K. Smith, E. Wood, and D. Maksimovic, “Modular approach for continuous cell-level balancing to improve performance of large battery packs,” *Energy Conversion Congress and Exposition (ECCE)*, pp. 4327–4334, 2014.

[70] R. Gogoana, *Internal resistance variances in lithium-ion batteries and implications in manufacturing*. Master’s thesis, Massachusetts Institute of Technology, USA, 2012.

[71] 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*, vol. 34, pp. 216 – 231, 2010.

[72] A. A. Pesaran, A. Vlahinos, and S. D. Burch, “Thermal Performance of EV and HEV Battery Modules and Packs,” in *International Electric Vehicle Symposium*, 1997.

[73] B. Y. Liaw, E. Roth, R. Jungst, G. Nagasubramanian, H. Case, and D. Doughty, “Correlation of Arrhenius behaviors in power and capacity fades with cell impedance and heat generation in cylindrical lithium-ion cells,” *Journal of Power Sources*, vol. 119-121, pp. 874–886, 2003.

[74] T. Bandhauer, S. Garimella, and T. Fuller, “A Critical Review of Thermal Issues in Lithium-Ion Batteries,” *Journal of The Electrochemical Society*, vol. 158, p. R1, 01 2011.

[75] H. Bergveld, W. S. Kruijt, and P. H. L. Notten, *Battery Management Systems : Design by Modelling*. Philips Research Book Series, Springer Netherlands, 2002.

[76] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee, “Systems Engineering for Industrial Cyber–Physical Systems Using Aspects,” *Proceedings of the IEEE*, vol. 104, no. 5, pp. 997–1012, 2016.

[77] M. Broy, “Engineering Cyber-Physical Systems: Challenges and Foundations,” in *Complex Systems Design & Management*, Springer, Berlin, Heidelberg, 2013.

[78] S. Ramesh, B. Vogel-Heuser, W. Chang, D. Roy, L. Zhang, and S. Chakraborty, “INVITED: Specification, verification and design of evolving automotive software,” in *Design Automation Conference (DAC)*, 2017.

[79] T. Heurung and S. Walz, “Designing and implementing architectures for distributed automotive E/E systems,” 2008. White Paper.

[80] T. Heurung, “Bridging Automotive Design Domains with the Latest in Functional Design Technology,” in *SAE Technical Paper*, 2015.

[81] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty, “Battery- and Aging-Aware Embedded Control Systems for Electric Vehicles,” in *Real-Time Systems Symposium (RTSS)*, 2014.

[82] A. Lamprecht, M. Baumann, T. Massier, and S. Steinhorst, “Decentralized Non-Neighbor Active Charge Balancing in Large Battery Packs,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[83] F. Baronti, R. Roncella, and R. Saletti, “Performance comparison of active balancing techniques for lithium-ion batteries,” *Journal of Power Sources*, vol. 267, pp. 603 – 609, 2014.

[84] 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,” *Transaction on Design Automation of Electronic Systems*, vol. 21, pp. 62:1–62:26, June 2016.

[85] S. Narayanaswamy, M. Kauer, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "Modular Active Charge Balancing for Scalable Battery Packs," *Transaction on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 974–987, March 2017.

[86] D. Roy, M. Balszun, T. Heurung, S. Chakraborty, and A. Naik, "Waterfall is too slow, let's go Agile: Multi-domain Coupling for Synthesizing Automotive Cyber-Physical Systems," in *International Conference on Computer-Aided Design (ICCAD)*, 2018.

[87] D. Roy, M. Balszun, T. Heurung, and S. Chakraborty, "Multi-Domain Coupling for Automated Synthesis of Distributed Cyber-Physical Systems," in *International Symposium on Circuits and Systems (ISCAS)*, 2018.

[88] A. Rajhans, A. Bhave, I. Ruchkin, B. H. Krogh, D. Garlan, A. Platzer, and B. Schmerl, "Supporting Heterogeneity in Cyber-Physical Systems Architectures," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3178–3193, 2014.

[89] J. Sztipanovits, "Model Integration and Cyber Physical Systems: A Semantics Perspective," in *Formal Methods* (M. Butler and W. Schulte, eds.), p. 1, Springer Berlin Heidelberg, 2011.

[90] P. J. Antsaklis, X. D. Koutsoukos, and J. Zaytoon, "On Hybrid Control of Complex Systems: A Survey," *European Journal of Automation*, vol. 32, no. 9-10, pp. 1023–1045, 1998.

[91] B. De Schutter, W. P. M. H. Heemels, J. Lunze, and C. Prieur, *Survey of modeling, analysis, and control of hybrid systems.* Handbook of Hybrid Systems Control – Theory, Tools, Applications, Cambridge University Press, 2009.

[92] R. Alur, "Formal Verification of Hybrid Systems," in *International conference on Embedded Software (EMSOFT)*, 2011.

[93] D. Roy, C. Hobbs, J. H. Anderson, M. Caccamo, and S. Chakraborty, "Timing Debugging for Cyber-Physical Systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[94] J. Sztipanovits, T. Bapty, S. Neema, X. Koutsoukos, and E. Jackson, "Design tool chain for cyber-physical systems: Lessons learned," in *Design Automation Conference (DAC)*, 2015.

[95] B. C. Kuo, *Digital Control Systems.* HRW Series in Electrical and Computer Engineering, Holt, Rinehart and Winston, Inc., 1980.

[96] W. C. Schultz and V. C. Rideout, "Control System Performance Measures: Past, Present, and Future," *Transactions on Automatic Control*, vol. AC-6, no. 1, pp. 22–35, 1961.

[97] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems (3rd ed.).* Prentice Hall Information and System Sciences, Prentice-Hall Inc., 1997.

[98] F. Sagstetter, P. Waszecki, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "Multischedule Synthesis for Variant Management in Automotive Time-Triggered Systems," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 637–650, 2016.

[99] F. Sagstetter, M. Lukasiewycz, and S. Chakraborty, "Generalized Asynchronous Time-Triggered Scheduling for FlexRay," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 2, pp. 214–226, 2017.

[100] S. S. Craciunas and R. S. Oliver, "SMT-based task-and network-level static schedule generation for time-triggered networked systems," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2014.

[101] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task-and network-level schedule co-synthesis of Ethernet-based time-triggered systems," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[102] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[103] J. Xu and D. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations," *IEEE Transactions on Software Engineering*, vol. 16, no. 3, pp. 360–369, 1990.

[104] R. J. Bril, *Real-time scheduling for media processing using conditionally guaranteed budgets*. Dissertation, Technische Universiteit Eindhoven, Eindhoven, 2004.

[105] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, no. 1-3, pp. 63–119, 2009.

[106] R. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[107] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing Analysis of the FlexRay Communication Protocol," *Real-Time Systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[108] H. Zeng, A. Ghosal, and M. Di Natale, "Timing Analysis and Optimization of FlexRay Dynamic Segment," in *International Conference on Computer and Information Technology (CIT)*, 2010.

[109] C. Hobbs, D. Roy, P. S. Duggirala, F. D. Smith, S. Samii, J. H. Anderson, and S. Chakraborty, "Perception Computing-Aware Controller Synthesis for Autonomous Systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[110] A. Cervin, “Stability and worst-case performance analysis of sampled-data control systems with input and output jitter,” in *American Control Conference (ACC)*, 2012.

[111] S. Ghosh, A. Mondal, D. Roy, P. H. Kindt, S. Dey, and S. Chakraborty, “Proactive feedback for networked cps,” in *ACM Symposium on Applied Computing*, 2021.

[112] E. P. van Horssen, A. R. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. P. M. H. Heemels, “Performance analysis and controller improvement for linear systems with (m, k)-firm data losses,” in *European Control Conference (ECC)*, 2016.

[113] W. Geelen, D. Antunes, J. P. M. Voeten, R. R. H. Schiffelers, and W. P. M. H. Heemels, “The Impact of Deadline Misses on the Control Performance of High-End Motion Control Systems,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1218–1229, 2016.

[114] D. Antunes and W. P. M. H. Heemels, “Frequency-Domain Analysis of Control Loops With Intermittent Data Losses,” *IEEE Transactions of Automatic Control*, vol. 61, no. 8, pp. 2295–2300, 2016.

[115] I. Saha, S. Baruah, and R. Majumdar, “Dynamic scheduling for networked control systems,” in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2015.

[116] R. Majumdar, I. Saha, and M. Zamani, “Performance-aware scheduler synthesis for control systems,” in *International conference on Embedded software (EMSOFT)*, 2011.

[117] D. Goswami, R. Schneider, and S. Chakraborty, “Relaxing Signal Delay Constraints in Distributed Embedded Controllers,” *Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2337–2345, 2014.

[118] W. Chang, D. Goswami, S. Chakraborty, and A. Hamann, “OS-Aware Automotive Controller Design Using Non-Uniform Sampling,” *Transaction on Cyber Physical Systems*, vol. 2, no. 4, 2018.

[119] A. Podelski and S. Wagner, “Model Checking of Hybrid Systems: From Reachability Towards Stability,” in *International Conference on Hybrid System: Computation and Control (HSCC)*, 2006.

[120] A. Podelski and S. Wagner, “Region Stability Proofs for Hybrid Systems,” in *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2007.

[121] A. Anta, R. Majumdar, I. Saha, and P. Tabuada, “Automatic verification of control system implementations,” in *International conference on Embedded software (EMSOFT)*, 2010.

[122] E. Feron, “From Control Systems to Control Software,” *IEEE Control Systems Magazine*, vol. 30, no. 6, pp. 50–71, 2010.

[123] R. Majumdar, I. Saha, and M. Zamani, “Synthesis of minimal-error control software,” in *International conference on Embedded software (EMSOFT)*, 2012.

[124] G. M. Mancuso, E. Bini, and G. Pannocchia, “Optimal Priority Assignment to Control Tasks,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, pp. 161:1–161:17, 2014.

[125] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, “Stability-aware analysis and design of embedded control systems,” in *International Conference on Embedded Software (EMSOFT)*, 2013.

[126] A. Minaeva, D. Roy, B. Akesson, Z. Hanzálek, and S. Chakraborty, “Control Performance Optimization for Application Integration on Automotive Architectures,” *IEEE Transactions on Computers*, vol. 70, no. 7, pp. 1059–1073, 2021.

[127] A. Aminifar, E. Bini, P. Eles, and Z. Peng, “Designing Bandwidth-Efficient Stabilizing Control Servers,” in *Real-Time Systems Symposium (RTSS)*, 2013.

[128] A. Aminifar, E. Bini, P. Eles, and Z. Peng, “Analysis and Design of Real-Time Servers for Control Applications,” *Transactions on Computers*, vol. 65, no. 3, pp. 834–846, 2016.

[129] Y. Wu, G. Buttazzo, E. Bini, and A. Cervin, “Parameter Selection for Real-Time Controllers in Resource-Constrained Systems,” *Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 610–620, 2010.

[130] R. Schneider, D. Goswami, A. Masrur, and S. Chakraborty, “QoC-oriented efficient schedule synthesis for mixed-criticality cyber-physical systems,” in *Forum on Specification and Design Languages (FDL)*, 2012.

[131] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty, “Multi-layered scheduling of mixed-criticality cyber-physical systems,” *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1215–1230, 2013.

[132] 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 *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013.

[133] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. Annaswamy, “Fault-tolerant control synthesis and verification of distributed embedded systems,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2014.

[134] A. R. B. Behrouzian, D. Goswami, M. Geilen, M. Hendriks, H. A. Ara, E. P. van Horssen, W. P. M. H. Heemels, and T. Basten, “Sample-drop firmness analysis of TDMA-scheduled control applications,” in *Symposium on Industrial Embedded Systems (SIES)*, 2016.

[135] P. Tabuada, “Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks,” *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[136] A. Anta and P. Tabuada, “Self-triggered stabilization of homogeneous control systems,” in *American Control Conference (ACC)*, 2008.

[137] M. Velasco, P. Martí, and E. Bini, “On Lyapunov sampling for event-driven controllers,” in *Conference on Decision and Control (CDC)*, 2009.

[138] R. Postoyan, P. Tabuada, D. Nešić, and A. Anta, “Event-triggered and self-triggered stabilization of distributed networked control systems,” in *Conference on Decision and Control and European Control Conference (CDC-ECC)*, 2011.

[139] M. Abdelrahim, V. S. Dolk, and W. P. M. H. Heemels, “Input-to-state stabilizing event-triggered control for linear systems with output quantization,” in *Conference on Decision and Control (CDC)*, 2016.

[140] M. Velasco, P. Martí, and E. Bini, “Control-Driven Tasks: Modeling and Analysis,” in *Real-Time Systems Symposium (RTSS)*, 2008.

[141] A. Aminifar, P. Tabuada, P. Eles, and Z. Peng, “Self-triggered controllers and hard real-time guarantees,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[142] M. Balszun, D. Roy, L. Zhang, W. Chang, and S. Chakraborty, “Effectively utilizing elastic resources in networked control systems,” in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[143] D. Roy, M. Balszun, D. Goswami, and S. Chakraborty, “Hybrid Automotive In-Vehicle Networks,” in *IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2017.

[144] D. Goswami, R. Schneider, and S. Chakraborty, “Re-engineering cyber-physical control applications for hybrid communication protocols,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011.

[145] A. Masrur, D. Goswami, R. Schneider, H. Voit, A. Annaswamy, and S. Chakraborty, “Schedulability analysis of distributed cyber-physical applications on mixed time-/event-triggered bus architectures with retransmissions,” in *International Symposium on Industrial and Embedded Systems (SIES)*, 2011.

[146] A. Masrur, D. Goswami, S. Chakraborty, J. Chen, A. Annaswamy, and A. Banerjee, “Timing analysis of cyber-physical applications for hybrid communication protocols,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[147] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin, “Designing high-quality embedded control systems with guaranteed stability,” in *Real-Time Systems Symposium (RTSS)*, 2012.

[148] A. Aminifar, E. Bini, P. Eles, and Z. Peng, “Bandwidth-efficient controller-server co-design with stability guarantees,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014.

[149] S. Samii, A. Cervin, P. Eles, and Z. Peng, “Integrated scheduling and synthesis of control applications on distributed embedded systems,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009.

[150] S. Samii, P. Eles, Z. Peng, and A. Cervin, “Design optimization and synthesis of FlexRay parameters for embedded control applications,” in *International Symposium on Electronic Design, Test and Application (DELTA)*, 2011.

[151] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, “Control-quality driven design of cyber-physical systems with robustness guarantees,” in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2013.

[152] W. Chang, D. Roy, X. S. Hu, and S. Chakraborty, “Cache-aware task scheduling for maximizing control performance,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[153] M. Glaß, M. Lukasiewyc, J. Teich, U. D. Bordoloi, and S. Chakraborty, “Designing heterogeneous ECU networks via compact architecture encoding and hybrid timing analysis,” in *Design Automation Conference (DAC)*, 2009.

[154] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, “A Real-Time Scratchpad-Centric OS for Multi-Core Embedded Systems,” in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[155] W. Chang, D. Roy, S. Zhao, A. Annaswamy, and S. Chakraborty, “CPS-oriented Modeling and Control of Traffic Signals Using Adaptive Back Pressure,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[156] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & Systems*. Prentice Hall, New Jersey, USA, 1997.

[157] W. Chang, L. Zhang, D. Roy, and S. Chakraborty, *Control/architecture codesign for cyber-physical systems*. Handbook of Hardware/Software Codesign, Springer Netherlands, 2017.

[158] C.-T. Chen, *System and Signal Analysis*. The Oxford Series in Electrical and Computer Engineering, Oxford University Press, UK, 1994.

[159] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation and Design*. Nob Hill Publishing, LLC, 2017.

[160] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer US, 2011.

[161] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty, “Constraint-driven synthesis and tool-support for FlexRay-Based automotive control systems,” in *International conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, 2011.

[162] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, and et al., “The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools,” *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, 2008.

[163] W. C. Lee, D. Drury, and P. Mellor, “Comparison of passive cell balancing and active cell balancing for automotive batteries,” in *Vehicle Power and Propulsion Conference (VPPC)*, 2011.

[164] T. H. Phung, A. Collet, and J. Crebier, “An Optimized Topology for Next-to-Next Balancing of Series-Connected Lithium-ion Cells,” *Transactions on Power Electronics*, vol. 29, no. 9, pp. 4603–4613, 2014.

[165] S. Narayanaswamy, S. Park, S. Steinhorst, and S. Chakraborty, “Multi-Pattern Active Cell Balancing Architecture and Equalization Strategy for Battery Packs,” in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2018.

[166] S. Manias, *Power Electronics and Motor Drive Systems*. Academic Press, 2016.

[167] M. Kauer, S. Narayanaswamy, M. Lukasiewycz, S. Steinhorst, and S. Chakraborty, “Inductor optimization for active cell balancing using geometric programming,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[168] M. Kauer, S. Narayanaswamy, S. Steinhorst, and S. Chakraborty, “Rapid Analysis of Active Cell Balancing Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 4, pp. 694–698, 2017.

[169] R. G. Powell, *Introduction to Electric Circuits*. Essential Electronic Series, published by: Arnold (a member of the Hodder Headline Group), 1995.

[170] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, Belmont, Massachusetts, 1998.

[171] IBM, “IBM ILOG CPLEX Optimizer.” Online (accessed 15 Jan 2020). `https://www.ibm.com/analytics/cplex-optimizer`.

[172] Gurobi Optimization Version 9.0, “Gurobi Optimizer.” Reference Manual (accessed 15 Jan 2020). `https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.0/refman.pdf`.

[173] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *International Conference on Neural Networks (ICNN)*, 1995.

[174] W. Chang, *Resource-Aware Automotive Control Systems Design*. Dissertation, Technische Universität München, München, 2017.

[175] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*, pp. 305–343. Handbook of Model Checking, Springer International Publishing, Cham, 2018.

[176] A. Fröhlich, *Theoretical and Practical Aspects of Bit-Vector Reasoning*. Dissertation, Johannes Kepler Universität Linz, Linz, 2016.

[177] S. Falke, C. Sinz, and F. Merz, "A Theory of Arrays with set and copy Operations," in *International Workshop on Satisfiability Modulo Theories (SMT)*, 2013.

[178] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal, "BDD based procedures for a theory of equality with uninterpreted functions," in *Computer Aided Verification (CAV)* (A. J. Hu and M. Y. Vardi, eds.), vol. 1427 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 1998.

[179] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT Solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (N. Piterman and S. Smolka, eds.), vol. 7795 of *Lecture Notes in Computer Science*, Springer, 2013.

[180] K. Scheibler, "iSAT3 Manual." Reference Manual, 2016. `https://projects.informatik.uni-freiburg.de/attachments/download/671/isat3_manual-0.03-20161213.pdf` (accessed 15 Januar 2020).

[181] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, vol. 5, no. 7, p. 394–397, 1962.

[182] L. de Moura and N. Njørner, "Z3: An Efficient SMT Solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (C. R. Ramakrishnan and J. Rehof, eds.), vol. 4963 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2008.

[183] B. Dutertre, "Yices 2.2," in *Computer-Aided Verification (CAV)* (A. Biere and R. Bloem, eds.), vol. 8559 of *Lecture Notes in Computer Science*, pp. 737–744, Springer, 2014.

[184] G. Behrmann, A. David, and K. G. Larsen, "A Tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems (SFM-RT)* (M. Bernardo and F. Corradini, eds.), vol. 3185 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2004.

[185] A. David, J. I. Rasmussen, K. G. Larsen, and A. Skou, "Model-based Framework for Schedulability Analysis Using Uppaal 4.1," 2009.

[186] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 17, no. 4, pp. 397–415, 2015.

[187] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, and P. Mckenzie, *KRONOS — Model Checking of Real-time Systems*. Systems and Software Verification, Springer, Berlin, Heidelberg, 2001.

[188] G. Chartrand and P. Zhang, *Chromatic Graph Theory*. Chapman and Hall, London, 2008.

[189] F. Maffray, "On the coloration of perfect graphs," in *Recent Advances in Algorithms and Combinatorics* (B. A. Reed and C. L. Sales, eds.), ch. 3, pp. 65–84, Springer, Oxford, 2003.

[190] M. Habib, R. McConnell, C. Paul, and L. Viennot, "Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing," *Theoretical Computer Science*, vol. 234, no. 1-2, pp. 59–84, 2000.

[191] F. Gavril, "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 180–187, 1972.

[192] C. S. Wang and R. S. Chang, "Parallel Maximal Cliques Algorithms for Interval Graphs with Applications," in *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, 1994.

[193] C. Lekkeikerker and J. Boland, "Representation of a finite graph by a set of intervals on the real line," *Fundamenta Mathematicae*, vol. 51, no. 1, pp. 45–64, 1962.

[194] G. Christodoulou and V. Zissimopoulos, "ON-LINE MAXIMUM INDEPENDENT SET IN CHORDAL GRAPHS," *Foundations of Computing and Decision Sciences*, vol. 30, no. 4, 2005.

[195] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of FlexRay-based distributed control systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[196] P. Marti, R. Villa, J. M. Fuertes, and G. Fohler, "On real-time control tasks schedulability," in *European Control Conference (ECC)*, 2001.

[197] L. Ma, F. Xia, and Z. Peng, "Integrated Design and Implementation of Embedded Control Systems with Scilab," in *Sensors*, 2008.

[198] M. E. M. B. Gaid, A. Cela, and Y. Hamam, "Optimal integrated control and scheduling of networked control systems with communication constraints: application to a car suspension system," *Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 776–787, 2006.

[199] P. Marti, J. Fuertes, G. Fohler, and K. Ramamritham, "Improving quality-of-control using flexible timing constraints: metric and scheduling," in *Real-Time Systems Symposium (RTSS)*, 2002.

[200] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich, "FlexRay schedule optimization of the static segment," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009.

[201] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automtoive software," in *Design, Automation, and Test in Europe Conference & Exhibition (DATE)*, 2012.

[202] W. C. Messner and D. M. Tilbury, "Control tutorials for matlab and simulink: a web-based approach." `http://ctms.engin.umich.edu/CTMS`, 1998.

[203] J. Fox, R. Roberts, C. Baier, L. Ho, L. Lacraru, and B. Gombert, "Modeling and control of a single motor electronic wedge brake," in *SAE Technical Paper 2007-01-0866*, 2009.

[204] K. Osman, M. Rahmat, and M. Ahmad, "Modelling and controller design for a cruise control system," in *International Colloquium on Signal Processing and Its Applications (CSPA)*, 2009.

[205] A. Schedl, "Goals and Architectures of FlexRay at BMW," in *Vector FlexRay Symposium*, 2007.

[206] P. Castelpietra, Y. -Q. Song, F. Simonot-Lion, and M. Attia, "Analysis and simulation methods for performance evaluation of a multiple networked embedded architecture," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1251–1264, 2002.

[207] S. Kollmann, V. Pollex, K. Kempf, F. Slomka, M. Traub, T. T. Bone, and D. Ag, "Comparative Application of Real-Time Verification Methods to an Automotive Architecture," in *International Conference on Real-Time and Network Systems (RTNS)*, 2010.

[208] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A GA-based scheduling method for FlexRay systems," in *International Conference on Embedded Software (EMSOFT)*, 2005.

[209] K. Schmidt and E. G. Schmidt, "Message scheduling for the FlexRay protocol: The static segment," *Transactions on Vehicular Technology*, vol. 58, no. 5, pp. 2170–2179, 2009.

[210] M. Grenier, L. Havet, and N. Navet, "Configuring the communication on FlexRay – the case of the static segment," in *European Congress on Embedded Real Time Software (ERTS)*, 2008.

[211] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Extensible and scalable time triggered scheduling," in *International Conference on Application of Concurrency to System Design (ACSD)*, 2005.

[212] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment," *Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 1–17, 2011.

[213] J. Mao, Z. Wu, and X. Wu, "A TDMA scheduling scheme for many-to-one communications in wireless sensor networks," *Computer Communications*, vol. 30, no. 4, pp. 863–872, 2007.

[214] M. Elmusrati, H. Sallabi, and H. Koivo, "Applications of multi-objective opitmization techniques in radio resource scheduling of cellular communication systems," *Wireless Communications*, vol. 7, no. 1, pp. 343–353, 2008.

[215] D. Roy, L. Zhang, W. Chang, D. Goswami, B. Vogel-Heuser, and S. Chakraborty, "Tool Integration for Automated Synthesis of Distributed Embedded Controllers," *ACM Transactions on Cyber-Physical Systems (TCPS)*, 2021. (to appear).

[216] SIMTOOLS GmbH, "SIMTOOLS: Model-Based Design Tools for FlexRay-based Applications."

[217] SIMTOOLS GmbH, "SIMTARGET: C code generation from SIMTOOLS models for CAN, FlexRay, and I/O."

[218] C. Phagoo ang G. Freiberger and D. Millinger, "System Design Validation using Matlab/Simulink and EB Simtools at Ford Motor Company." White Paper, 2009.

[219] J. Hou and W. Wolf, "Process Partitioning for Distributed Embedded Systems," in *International Workshop on Hardware/Software Co-Design (CODES)*, 1996.

[220] C. -C. Shen and W. -H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," *IEEE Transactions on Computers*, vol. C-34, no. 3, pp. 197–203, 1985.

[221] "Data Model for ECU Network Systems," ASAM MCD-2 NET, Association for Standardization of Automation and Measuring Systems (ASAM), 2017.

[222] Q. Zhu and A. Sangiovanni-Vincentelli, "Codesign Methodologies and Tools for Cyber–Physical Systems," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1484–1500, 2018.

[223] Vector, "PREEvision: Model-Based Electric/Electronic Development." Online. `https://www.vector.com/de/en/products/products-a-z/software/preevision/`.

[224] Vector, "DaVinci Configurator Pro: Configure, Validate and Generate AUTOSAR Basic Software." Online. `https://www.vector.com/de/en/products/products-a-z/software/davinci-configurator-pro/`.

[225] Mentor, A Siemens Business, "Capital: Enabling the Electrical Model Based Enterprise." Online. `https://www.mentor.com/products/electrical-design-software/capital/`.

[226] Mentor, A Siemens Business, "VSTAR Tools." Online. `https://www.mentor.com/embedded-software/autosar/tools`.

[227] dSpace, “SystemDesk: Modeling system architecture and generating virtual ECUs.” Online. `https://www.dspace.com/en/ltd/home/products/sw/system_architecture_software/systemdesk.cfm#143_25611`.

[228] dSpace, “TargetLink: Production code generation for the highest demands.” Online. `https://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm`.

[229] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, “Metropolis: An Integrated Electronic System Design Environment,” *Computer*, vol. 36, no. 4, pp. 45–52, 2003.

[230] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, “MetroII: A Design Environment for Cyber-Physical Systems,” *Transactions on Embedded Computing Systems*, vol. 12, no. 1s, 2013.

[231] J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong, “Taming heterogeneity - the Ptolemy approach,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[232] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. Sangiovanni-Vincentelli, and E. A. Lee, “Metronomy: A function-architecture co-simulation framework for timing verification of cyber-physical systems,” in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014.

[233] Y. Yang, A. Pinto, A. Sangiovanni-Vincentelli, and Q. Zhu, “A Design Flow for Building Automation and Control Systems,” in *Real-Time Systems Symposium (RTSS)*, 2010.

[234] F. Balarin, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, Y. Watanabe, and Guang Yang, “Concurrent execution semantics and sequential simulation algorithms for the Metropolis meta-model,” in *International Symposium on Hardware/Software Codesign (CODES)*, 2002.

[235] D. Roy, W. Chang, S. K. Mitter, and S. Chakraborty, “Tighter Dimensioning of Heterogeneous Multi-Resource Autonomous CPS with Control Performance Guarantees,” in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[236] D. Roy, S. Ghosh, Q. Zhu, M. Caccamo, and S. Chakraborty, “GoodSpread: Criticality-Aware Static Scheduling of CPS with Multi-QoS Resources,” in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[237] D. Majumdar, L. Zhang, P. Bhaduri, and S. Chakraborty, “Reconfigurable communication middleware for FlexRay-based distributed embedded systems,” in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015.

[238] A. David, J. Illum, K. G. Larsen, and A. Skou, “Model-based framework for schedulability analysis using UPPAAL 4.1. Model-Based Design for Embedded Systems,” 2009.

[239] Z. Gu, Z. Wang, H. Chen, and H. Cai, “A model-checking approach to schedulability analysis of global multiprocessor scheduling with fixed offsets,” *International Journal of Embedded Systems*, vol. 6, pp. 176–187, 2014.

[240] H. Lin and P. J. Antsaklis, “Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results,” *IEEE Transactions on Automatic Control*, vol. 54, pp. 308–322, Feb. 2009.

[241] N. Thomas and P. Poongodi, “Position Control of DC Motor Using Genetic Algorithm Based PID Controller,” in *World Congress on Engineering (WCE)*, 2009.

[242] R. Schneider, D. Goswami, S. Chakraborty, U. Bordoloi, P. Eles, and Z. Peng, “On the Quantification of Sustainability and Extensibility of FlexRay Schedules,” in *Design Automation Conference (DAC)*, 2011.

[243] Q. Zhu, H. Liang, L. Zhang, D. Roy, W. Li, and S. Chakraborty, “Extensibility-driven automotive in-vehicle architecture design: Invited,” in *Design Automation Conference (DAC)*, 2017.

[244] L. Maldonado, W. Chang, D. Roy, A. Annaswamy, D. Goswami, and S. Chakraborty, “Exploiting System Dynamics for Resource-Efficient Automotive CPS Design,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[245] X. Ge, F. Yang, and Q. Han, “Distributed Networked Control Systems: A Brief Overview,” *Information Sciences*, vol. 380, pp. 117–131, 2017.

[246] W. Zhang, M. Branicky, and S. Phillips, “Stability of networked control systems,” *Control Systems*, vol. 21, no. 1, pp. 84–99, 2001.

[247] L. Zhang, D. Roy, P. Mundhenk, and S. Chakraborty, “Schedule Management Framework for Cloud-Based Future Automotive Software Systems,” in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.

[248] P. Mundhenk, G. Tibba, L. Zhang, F. Reimann, D. Roy, and S. Chakraborty, “Dynamic platforms for uncertainty management in future automotive E/E architectures,” in *Design Automation Conference (DAC)*, 2017.

[249] W. Chang, D. Roy, L. Zhang, and S. Chakraborty, “Model-based design of resource-efficient automotive control software,” in *International Conference on Computer-Aided Design (ICCAD)*, 2016.

[250] D. Roy, L. Zhang, W. Chang, and S. Chakraborty, “Automated synthesis of cyber-physical systems from joint controller/architecture specifications,” in *Forum on Specification and Design Languages (FDL)*, 2016.

[251] D. Roy, S. Narayanaswamy, A. Pr"obstl, and S. Chakraborty, “Multi-Stage Optimization for Energy-Efficient Active Cell Balancing in Battery Packs,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[252] D. Roy, S. Narayanaswamy, A. Pr"obstl, and S. Chakraborty, “Optimal Scheduling for Active Cell Balancing,” in *IEEE Real-Time Systems Symposium (RTSS)*, 2019.

[253] A. C.Baughman and M. Ferdowsi, “Double-Tiered Switched-Capacitor Battery Charge Equalization Technique,” *IEEE Transactions on Industrial Electronics*, vol. 55, no. 6, pp. 2277–2285, 2008.

[254] R. Fukui and H. Koizumi, “Double-tiered switched capacitor battery charge equalizer with chain structure,” in *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2013.

[255] M. Kauer, S. Narayanaswamy, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, “Many-to-many active cell balancing strategy design,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015.

[256] M. Kauer, S. Naranayaswami, S. Steinhorst, M. Lukasiewycz, S. Chakraborty, and L. Hedrich, “Modular system-level architecture for concurrent cell balancing,” in *Design Automation Conference (DAC)*, 2013.

[257] 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*, vol. 60, no. 6, pp. 2448–2457, 2011.

[258] C. Danielson, F. Borrelli, D. Oliver, D. Anderson, M. Kuang, and T. Phillips, “Balancing of battery networks via constrained optimal control,” in *American Control Conference (ACC)*, 2012.

[259] C. Danielson, F. Borrelli, D. Oliver, D. Anderson, and T. Phillips, “Constrained Flow Control in Storage Networks: Capacity Maximization and Balancing,” *Automatica*, vol. 49, no. 9, pp. 2612–2621, 2013.

[260] R. K. Ahuja, “Minimax linear programming problem,” *Operation Research Letters*, vol. 4, no. 3, pp. 131–134, 1985.

[261] J. Cao, N. Schofield, and A. Emadi, “Battery balancing methods: A comprehensive review,” in *Vehicle Power and Propulsion Conference (VPPC)*, 2008.

[262] M. Daowd, N. Omar, P. Van den Bossche, and J. Van Mierlo, “Passive and active battery balancing comparison based on matlab simulation,” in *Vehicle Power and Propulsion Conference (VPPC)*, 2011.

[263] S. Steinhorst, M. Lukasiewycz, S. Narayanaswamy, M. Kauer, and S. Chakraborty, “Smart Cells for Embedded Battery Management,” in *International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2014.

[264] L. He, L. Gu, L. Kong, Y. Gu, C. Liu, and T. He, “Exploring Adaptive Reconfiguration to Optimize Energy Efficiency in Large-Scale Battery Systems,” in *Real-Time Systems Symposium (RTSS)*, 2013.

[265] H. Kim and K. G. Shin, “On Dynamic Reconfiguration of a Large-Scale Battery System,” in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[266] H. Kim and K. G. Shin, “Scheduling of Battery Charge, Discharge, and Rest,” in *Real-Time Systems Symposium (RTSS)*, 2009.

[267] E. Kim, J. Lee, and K. G. Shin, “Modeling and Real-Time Scheduling of Large-Scale Batteries for Maximizing Performance,” in *Real-Time Systems Symposium (RTSS)*, 2015.

[268] E. Kim, K. G. Shin, and J. Lee, “Real-Time Discharge/Charge Rate Management for Hybrid Energy Storage in Electric Vehicles,” in *Real-Time Systems Symposium (RTSS)*, 2014.

[269] H. Heinimäki, P. Niska, and J. Ruutu, “Effect of Battery Charge on Energy Consumption,” in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[270] F. Kong, Q. Xiang, L. Kong, and X. Liu, “On-Line Event-Driven Scheduling for Electric Vehicle Charging via Park-and-Charge,” in *Real-Time Systems Symposium (RTSS)*, 2016.

[271] M. S. Branicky, V. S. Borkar, and S. K. Mitter, “A unified framework for hybrid control: model and optimal control theory,” *IEEE Transactions on Automatic Control*, vol. 43, no. 1, pp. 31–45, 1998.

[272] Y. Wang, X. Sun, P. Shi, and J. Zhao, “Input-to-State Stability of Switched Nonlinear Systems With Time Delays Under Asynchronous Switching,” *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 2261–2265, 2013.

[273] Y. Wang, X. Sun, and B. Wu, “Lyapunov–Krasovskii functionals for input-to-state stability of switched non-linear systems with time-varying input delay,” *IET Control Theory & Applications*, vol. 9, no. 11, pp. 1717–1722, 2015.

[274] W. P. M. H. Heemels, D. P. Borgers, N. van de Wouw, D. Nešić, and A. R. Teel, “Stability analysis of nonlinear networked control systems with asynchronous communication: A small-gain approach,” in *Conference on Decision and Control (CDC)*, 2013.

[275] Y. Wang, M. Xia, V. Gupta, and P. J. Antsaklis, “On feedback passivity of discrete-time nonlinear networked control systems with packet drops,” *IEEE Transactions on Automatic Control*, vol. 60, no. 9, pp. 2434–2439, 2014.

[276] J. Lei and H. K. Khalil, “Feedback Linearization for Nonlinear Systems With Time-Varying Input and Output Delays by Using High-Gain Predictors,” *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2262–2268, 2016.

[277] T. Takagi and M. Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 1, pp. 116–132, 1985.

[278] X. Zhang, G. Lu, and Y. Zheng, “Stabilization of networked stochastic time-delay fuzzy systems with data dropout,” *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 3, pp. 798—-807, 2008.

[279] H. Gao, Y. Zhao, and T. Chen, “$\mathcal{H}_\infty$ fuzzy control of nonlinear systems under unreliable communication links,” *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 2, pp. 265—-278, 2009.

[280] H. Li, C. Wu, and Z. Feng, “Fuzzy dynamic output-feedback control of non-linear networked discrete-time system with missing measurements,” *IET Control Theory & Applications*, vol. 9, no. 3, pp. 327—-335, 2015.

[281] J. Qiu, G. Feng, and H. Gao, “Fuzzy-model-based piecewise $\mathcal{H}_\infty$ static-output-feedback controller design for networked nonlinear systems,” *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 5, pp. 919—-934, 2010.

[282] D. Du, “Reliable $\mathcal{H}_\infty$ control for Takagi-Sugeno fuzzy systems with intermittent measurements,” *Nonlinear Analysis: Hybrid Systems*, vol. 6, no. 4, pp. 930—-941, 2012.

[283] Y. Zhao, H. Gao, and T. Chen, “Fuzzy constrained predictive control of nonlinear systems with packet dropouts,” *IET Control Theory & Applications*, vol. 4, no. 9, pp. 1665—-1677, 2010.

[284] H. Zhang, J. Yang, and C. Su, “T-S fuzzy-model-based robust $\mathcal{H}_\infty$ design for networked control systems with uncertainties,” *IEEE Transactions on Industrial Informatics*, vol. 3, no. 4, pp. 289—-301, 2007.

[285] H. Zhang, D. Yang, and T. Chai, “Guaranteed cost networked control for T-S fuzzy systems with time delays,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 2, pp. 160—-172, 2007.

[286] C. Peng and T. C. Yang, “Communication-delay-distribution-dependent networked control for a class of T-S fuzzy systems,” *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 2, pp. 326—-335, 2010.

[287] E. Tian, D. Yue, and Z. Gu, “Robust $\mathcal{H}_\infty$ control for nonlinear systems over network: A piecewise analysis method,” *Fuzzy Sets and Systems*, vol. 161, no. 21, pp. 2731—-2745, 2010.

[288] M. S. Mahmoud and A. A. Saif, “Robust quantized approach to fuzzy networked control systems,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 1, pp. 71—-81, 2012.

[289] J. Yan, Y. Xia, and L. Li, “Stabilization of fuzzy systems with quantization and packet dropouts,” *International Journal of Robust and Nonlinear Control*, vol. 24, no. 10, pp. 1563—-1583, 2014.

[290] T. E. Gibson, A. M. Annaswamy, and E. Lavretsky, “Improved transient response in adaptive control using projection algorithms and closed loop reference models,” in *AIAA Guidance Navigation and Control Conference*, 2012.

[291] T. E. Gibson, A. M. Annaswamy, and E. Lavretsky, “Closed–loop Reference Model Adaptive Control: Composite control and Observer Feedback,” in *IFAC International Workshop on Adaptation and Learning in Control and Signal Processing*, 2013.

[292] T. E. Gibson, A. M. Annaswamy, and E. Lavretsky, “Adaptive systems with closed-loop reference-models, Part I: Transient performance,” in *American Control Conference (ACC)*, 2013.

[293] T. E. Gibson, A. M. Annaswamy, and E. Lavretsky, “Closed-loop reference models for output-feedback adaptive systems,” in *European Control Conference (ECC)*, 2013.

[294] L. Chunmao and X. Jian, “Adaptive Delay Estimation and Control of Networked Control systems,” in *International Symposium on Communications and Information Technologies (ISCIT)*, 2006.

[295] H. Voit and A. Annaswamy, “Adaptive control of a Networked Control System with hierarchical scheduling,” in *American Control Conference (ACC)*, 2011.

[296] H. Voit, A. M. Annaswamy, R. Schneider, D. Goswami, and S. Chakraborty, “Adaptive switching controllers for systems with hybrid communication protocols,” in *American Control Conference (ACC)*, 2012.

[297] D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke, “Temporal isolation on multiprocessing architectures,” in *Design Automation Conference (DAC)*, 2011.

[298] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, “Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability,” *IEEE Micro*, vol. 30, no. 5, pp. 66–75, 2010.

[299] S. Girbal, X. Jean, J. L. Rhun, D. G. Pérez, and M. Gatti, “Deterministic platform software for hard real-time systems using multi-core COTS,” in *Digital Avionics Systems Conference (DASC)*, 2015.

[300] B. Zheng, P. Deng, R. Anguluri, Q. Zhu, and F. Pasqualetti, “Cross-Layer Codesign for Secure Cyber-Physical Systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 699–711, 2016.

[301] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu, “Security-Driven Codesign with Weakly-Hard Constraints for Real-Time Embedded Systems,” in *International Conference on Computer Design (ICCD)*, 2019.

[302] K. Vatanparvar and M. A. Al Faruque, “Battery lifetime-aware automotive climate control for electric vehicles,” in *Design Automation Conference (DAC)*, 2015.

[303] D. Goswami, D. Müller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty, “Fault-tolerant embedded control systems for unreliable hardware,” in *International Symposium on Integrated Circuits (ISIC)*, 2014.

List of Tables

List of Figures

List of Symbols

Variables	**Descriptions**
x	state of the plant
u	control input applied to the plant
y	output of the plant
A	continuous-time state matrix
B	continuous-time input matrix
C	continuous-time or discrete-time output matrix
h	sampling period
ϕ	discrete-time state matrix
Γ	discrete-time input matrix
d	delay between sensing (or sampling) and actuation
x_a	augmented state of the plant
ϕ_a	augmented discrete-time state matrix
Γ_a	augmented discrete-time input matrix
C_a	augmented discrete-time output matrix
λ_i	a system pole or a eigenvalue of the state-stransition matrix
K	feedback control gain
F	feedforward control gain
r	reference input for the plant
ϕ_{cl}	closed-loop state-transition matrix
γ	controllability matrix
$H(s)$	characteristic polynomial of the closed-loop system
T_i	a software task
o_i	offset of the task T_i
p_i	period of the task T_i
e_i	(worst-case) execution time of the task T_i
$\widehat{t}(T_i, k)$	start time of the k-th instance of the task T_i
$\tilde{t}(T_i, k)$	finish time of the k-th instance of the task T_i
N_s	number of slots in the FlexRay static segment
N_d	number of minislots in the FlexRay dynamic segment
Δ	length of a static slot in the FlexRay static segment
δ	length of a minislot in the FlexRay dynamic segment
m_i	a message

Variables	Descriptions
T_{bus}	bus cycle time
s_i	slot number assigned to a message m_i
b_i	base cycle assigned to a message m_i
r_i	repetition rate assigned to a message m_i
$\widehat{t}(m_i, k)$	start time of the k-th instance of the message m_i
$\tilde{t}(m_i, k)$	finish time of the k-th instance of the message m_i
$\mathcal{B}_i$	a cell in the battery pack
T^C	the time duration for a single charge transfer cycle
T_{ON}	the time duration for the charging phase in a charge transfer cycle
T_{OFF}	the time duration for the discharging phase in a charge transfer cycle
I_{peak}	the peak balancing current
V_α	the voltage across the cell $\mathcal{B}_\alpha$
R_α	total resistance in the current flow path during the charging phase
R_β	total resistance in the current flow path during the discharging phase
L	inductance of the inductor (or transformer)
Q_{tx}	amount of charge transferred from the source cell in a charge transfer cycle
Q_{rx}	amount of charge received by the destination cell in a charge transfer cycle
$\check{d}$	number of cells between the source cell and the destination cell
E_i	an ECU
$\mathcal{E}$	set of all ECUs in the system
$\mathcal{C}_i$	a control application
$\mathbb{C}$	set of all control applications in the system
e_{com}	execution time of a communication task
$\{A_i, B_i, C_i\}$	state, input, and output matrices of an application $\mathcal{C}_i$
h_i	sampling period of an application $\mathcal{C}_i$
d_i	delay of an application $\mathcal{C}_i$
K_i	(worst-case) feedback control gain of an application $\mathcal{C}_i$
F_i	start time of the feedforward control gain of an application $\mathcal{C}_i$
$T_s(T_{s,i})$	sensor task of an application ($\mathcal{C}_i$)
$T_c(T_{c,i})$	controller task of an application ($\mathcal{C}_i$)
$T_a(T_{a,i})$	actuator task of an application ($\mathcal{C}_i$)
$m_{s,i}$	sensor message of an application $\mathcal{C}_i$
$m_{c,i}$	control message of an application $\mathcal{C}_i$
N_{com}	number of configurable communication cycles
U	resource usage, i.e., no. of slots used in N_{com} consecutive cycles
$J(J_i)$	control performance (of an application $\mathcal{C}_i$)
J_i^n	normalized control performance of an application $\mathcal{C}_i$
J_i^n	required control performance of an application $\mathcal{C}_i$
J_{av}	average control performance of the system
tt	task type, i.e., sensor, controller, or actuator task

Variables	**Descriptions**
mt	message type, i.e., sensor or control message
$\{o_{tt,i}, p_{tt,i}, e_{tt,i}\}$	offset, period, and execution time of task type tt in an application $\mathcal{C}_i$
$\{s_{tt,i}, b_{tt,i}, r_{tt,i}\}$	slot number, base cycle, and repetition rate of message type tt in an application $\mathcal{C}_i$
par_i^s	schedule parameters of an application $\mathcal{C}_i$
par_i^c	control parameters of an application $\mathcal{C}_i$
par_i	parameters of an application $\mathcal{C}_i$
$\mathcal{P}$	parameters of the whole system
$\mathcal{F}$	the Pareto front
H	set of feasible sampling periods
$h^{(k)}$	a feasible sampling period
N_h	total number of feasible sampling periods
K_i^{k*}	optimal feedback control gain for an application C_i at a sampling period $h^{(k)}$
F_i^{k*}	optimal feedforward control gain for an application C_i at a sampling period $h^{(k)}$
J_i^{k*}	optimal control performance for an application C_i at a sampling period $h^{(k)}$
$\mathcal{T}(E_j)$	set of tasks mapped on the ECU E_j
$\phi_{a,i}$	augmented discrete-time state matrix of an application $\mathcal{C}_i$
$\Gamma_{a,i}$	augmented discrete-time input matrix of an application $\mathcal{C}_i$
$C_{a,i}$	augmented discrete-time output matrix of an application $\mathcal{C}_i$
pt	a particle in PSO
pos_{pt}	the position of a particle pt
vel_{pt}	the velocity of a particle pt
fit_{pt}	the fitness value of a particle pt
$pbest_{pt}$	personal best solution obtained so far for a particle pt
$gbest$	global best solution obtained so far
γ_{syn}	time constant required for ECU synchronization
U^-	minimum possible resource usage
U^+	maximum possible resource usage
U^Δ	minimum difference between two possible values of resource usage
U^*	a value of resource usage
$\mathcal{P}^*$	a feasible parameter set
J_{av}^*	a feasible value of average control performance
$\mathcal{P}^{c*}$	a feasible set of control parameters
$\mathcal{P}^{s*}$	a feasible set of schedule parameters
u^*	maximum permissible value of control input
$n_R(n_{R,i})$	length of the periodic switching pattern for a bimodal controller ($\mathcal{C}_i$)
$n_T(n_{T,i})$	number of consecutive instance in high-cost mode in a periodic pattern for a bimodal controller ($\mathcal{C}_i$)

Variables	**Descriptions**
$n_w(n_{w,i})$	number of control instances for which a bimodal controller (C_i) is waiting in the low-cost mode after a disturbance
$n_T^-(n_{T,i}^-)$	minimum number of consecutive control instances for which a bimodal controller (C_i) must stay in the high-cost mode after switching from the low-cost mode
$n_T^+(n_{T,i}^+)$	maximum number of consecutive control instances for which a bimodal controller (C_i) can stay in the high-cost mode after switching from the low-cost mode
$n_w^*(n_{w,i}^*)$	maximum number of control instances for which a bimodal controller (C_i) can wait in the low-cost mode after a disturbance without violating its requirements
d_{HC}	delay in the high-cost mode for a bimodal controller
d_{LC}	delay in the low-cost mode for a bimodal controller
$K_{HC}(K_{HC,i})$	controller or control gain for the high-cost mode of a bimodal controller ($\mathcal{C}_i$)
$K_{LC}(K_{LC,i})$	controller or control gain for the low-cost mode of a bimodal controller ($\mathcal{C}_i$)
Φ_{HC}	closed-loop state-transition matrix for the high-cost mode of a bimodal controller
Φ_{LC}	closed-loop state-transition matrix for the low-cost mode of a bimodal controller
Γ_0	discrete-time input matrix for calculating the impact of the current control input
Γ_1	discrete-time input matrix for calculating the impact of the last control input
J_i^*	settling time requirement for a bimodal controller $\mathcal{C}_i$
$t_{a,i}$	disturbance interarrival time for the control application $\mathcal{C}_i$
$J_{HC,i}$	settling time when the bimodal controller $\mathcal{C}_i$ operates only in the high-cost mode
$J_{LC,i}$	settling time when the bimodal controller $\mathcal{C}_i$ operates only in the low-cost mode
n_{id}	number of slot ids used by the bimodal controllers
n_R^*	number of control cycles in the static schedule of bimodal controllers
$\mathbb{C}_{srt}$	sorted list of applications to which the first-fit heuristic is applied
$\pi_{i,j}$	a charge transfer pair where the cell $\mathcal{B}_i$ transfers charge to the cell $\mathcal{B}_j$
$c_{i,j}$	number of cycles for which the cell $\mathcal{B}_i$ transfers charge to the cell $\mathcal{B}_j$
N_B	number of cells in the battery pack
$\widehat{d}$	maximum number of cells between a charge transfer pair of cells
$\mathcal{P}_i$	the set of cells with which the cell $\mathcal{B}_i$ can pair for charge transfer
$\mathcal{CT}$	the set of all feasible charge transfer pair of cells
$Q_{tx}(i,j)$	charge transmitted by the source cell $\mathcal{B}_i$ when paired with the destination cell $\mathcal{B}_j$ in one charge transfer cycle

Variables	**Descriptions**
$Q_{rx}(j,i)$	charge received by the destination cell $\mathcal{B}_j$ when paired with the source cell $\mathcal{B}_i$ in one charge transfer cycle
$Q^{-}_{i,j}$	total charge transmitted by the source cell $\mathcal{B}_i$ when paired with the destination cell $\mathcal{B}_j$
$Q^{+}_{j,i}$	total charge received by the destination cell $\mathcal{B}_j$ when paired with the source cell $\mathcal{B}_i$
Q^{T}_{i}	total charge transmitted by the cell $\mathcal{B}_i$ during a charge equalization session
Q^{R}_{i}	total charge received by the cell $\mathcal{B}_i$ during a charge equalization session
$E_{tx}(i,j)$	energy transmitted by the source cell $\mathcal{B}_i$ when paired with the destination cell $\mathcal{B}_j$ in one charge transfer cycle
$E_{rx}(j,i)$	energy received by the destination cell $\mathcal{B}_j$ when paired with the source cell $\mathcal{B}_i$ in one charge transfer cycle
$E_{\Delta}(i,j)$	energy dissipated per cycle when $\mathcal{B}_i$ transfers charge to $\mathcal{B}_j$
$E_{i,j}$	total energy dissipation for charge transfer between $\mathcal{B}_i$ amd $\mathcal{B}_j$ during a balancing session
$Q_{i,f}$	final charge level of the cell $\mathcal{B}_i$ after balancing
$Q_{i,s}$	initial charge level of the cell $\mathcal{B}_i$ before balancing
Q_{th}	charge threshold for cell balancing, i.e., any two cells in the pack must be within the charge threshold for the battery pack to be balanced
E_D	total energy dissipation in the battery pack during a cell balancing session
$\mathcal{U}_{i,j}$	When $\mathcal{B}_i$ transfers charge to $\mathcal{B}_j$, the set of cells that become unavailable for additional charge transfers
$\mathcal{G}$	a conflict graph where the vertices are the charge transfer pairs and an edge between two vertices denote that the corresponding two pairs are not simultaneously schedulable
$\mathcal{E}_G$	the set of edges in the graph $\mathcal{G}$
$\{\pi_{a,b}, \pi_{c,d}\}$	an edge in the conflict graph represented by the unordered pair of vertices
$\pi^{p}_{a,b}$	a charge transfer cycle of the pair $\pi_{a,b}$
$\mathcal{G}_r$	a conflict graph where the vertices are the charge transfer pairs with non-zero charge transfer cycles and an edge between two vertices denote that the corresponding two pairs are not simultaneously schedulable
$\mathcal{G}_d$	a conflict graph where the vertices are the charge transfer cycles of feasible pairs of cells and an edge between two vertices denote that the corresponding two cycles cannot occur at the same time
N_G	minimum number of colors required to color all vertices in the graph $\mathcal{G}_d$
T^{*}_{Φ}	Balancing time corresponding to a charge transfer schedule
$I^{r}_{i,j}$	an interval on the real line representing the charge transfer cycle $\pi^{r}_{i,j}$

Variables	**Descriptions**
$\mathcal{CT}^+$	set of charge transfer pairs of cells with non-zero charge transfer cycles
$\mathcal{CP}_k$	set of charge transfer pairs that have scheduling conflicts with each other
$\mathcal{S}_{\mathcal{CP}}$	set of sets of conflicting charge transfer pairs
cl_k	a clique in the graph $\mathcal{G}_d$
$\mathcal{S}_c$	set of cliques in the graph $\mathcal{G}_d$
E_D^*	an upper bound on energy dissipation during cell balancing
$\widehat{T}_\Phi$	An upper bound on the balancing time
$\mathcal{O}_r$	a perfect elimination ordering of vertices in the graph $\mathcal{G}_r$
$\mathcal{O}_d$	a perfect elimination ordering of vertices in the graph $\mathcal{G}_d$

Abbreviations

ADAS advanced driver-assistance system

ASW application software

AUTOSAR AUTomotive Open Software ARchitecture

BCET best-case execution time

BMS battery management system

BSW basic software

CAN Controller Area Network

COTS commercial-off-the-shelf

CPS cyber-physical system

CQLF common quadratic Lyapunov function

CSMA/CD carrier-sense multiple access with collision detection

CSMA/CR carrier-sense multiple access/collision resolution

DMA direct memory access

DSE design space exploration

EES electrical energy storage

E/E electrical and electronic

ECU electronic control unit

EV electric vehicle

FTDMA flexible TDMA

HEV hybrid electric vehicle

HIL hardware-in-the-loop

ILP integer linear programming

Lex-BFS lexicographic breadth first search

Li-ion lithium ion

LP linear programming

LQG linear quadratic Guassian

LQR linear quadratic regulator

LTI linear and time-invariant

LTL linear temporal logic

MIL model-in-the-loop

MILP mixed-integer linear programming

MOSFET metal-oxide-semiconductor field-effect transistor

MOST Media-Oriented Systems Transport

MVC minimum vertex coloring

OEM original equipment manufacturer

OS operating system

OSEK Offene Systeme und deren Schnittstellen für Elektronik in Kraftfahrzeugen

PEOV perfect elimination ordering of the vertices

PSO particle swarm optimization

PWM pulse-width modulation

RTE runtime environment

RTOS real-time operating system

RTW Real-Time Workshop

SAT boolean satisfiability

SIL software-in-the-loop

SISO single-input single-output

SMT Satisfiability Modulo Theories

SoC state-of-charge

SoH state-of-health

TA timed automaton

TDMA time-division multiple access

TSN Time-Sensitive Networking

TTCAN time-triggered CAN

VDX Vehicle Distributed eXecutive

WCET worst-case execution time