Constraint-Driven Synthesis and Tool-Support for FlexRay-Based Automotive Control Systems

Reinhard Schneider, Dip Goswami*, Sohaib Zafar, Samarjit Chakraborty
TU Munich, Germany
{reinhard.schneider,dip.goswami,sohaib.zafar, samarjit.chakraborty}@rcs.ei.tum.de

Martin Lukasiewycz
TUM CREATE, Singapore
martin.lukasiewycz@tum-create.edu.sg

ABSTRACT

Emerging bus protocols such as FlexRay provide an expedient platform for the design of automotive control systems due to its high bandwidth and deterministic temporal behavior. However, the choice of suitable platform parameters such as task and message schedules becomes a challenging design problem as the protocol is complex in nature and enforces a tight coupling between local task schedules on ECUs and global bus schedules. Although there exist several commercial off-the-shelf (COTS) design tools for FlexRay and control systems, current tools do not provide any mechanism for automatically synthesizing the platform parameters from the controller specifications. In this work we synthesize controllers subject to specified control goals while taking into account platform-specific properties. In particular, we translate the *timing* constraints derived from the control design into platform constraints that need to be satisfied by the control-related tasks and messages. For this purpose, we formulate and solve a constraint satisfaction problem (CSP) to synthesize feasible platform parameters that can be realized by the underlying operating systems and the FlexRay bus. Our design flow may be easily integrated with existing FlexRay design tools and will significantly ease (and automate) the existing design process. We show the applicability of our results by implementing two automotive control systems on a Hardware-in-the-Loop (HiL) setup and study how different bus configurations affect the controller synthesis and the choice of platform parameters.

Categories and Subject Descriptors

C.3 [**Special-Purpose And Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Performance

*This author is an Alexander von Humboldt Research Fellow at TU Munich, Germany.

CODES+ISSS'11, October 9–14, 2011, Taipei, Taiwan.

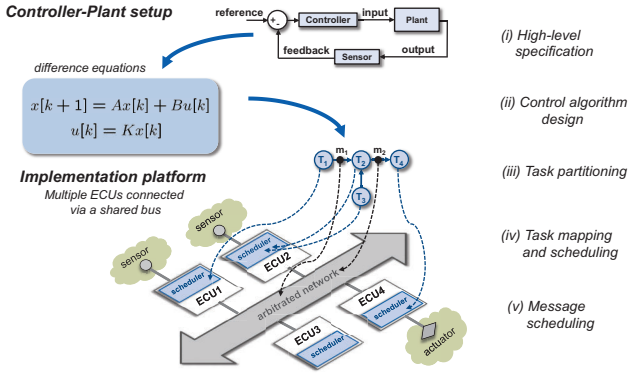

Figure 1: Conventional design flow.

Keywords

Automotive Control Systems, Cyber-Physical Systems, Schedule Synthesis, Operating Systems, FlexRay

1. INTRODUCTION

Today's automotive in-vehicle networks consist of up to 100 Electronic Control Units (ECUs), sensors, controllers and actuators that are connected via arbitrated, shared buses such as Ethernet, CAN, or FlexRay. In the context of such setups, communication and signal processing of control applications require a tight conjoining and coordination between computational (cyber) units and physical resources. Such setups are generally referred as cyber-physical systems [1]. Fig. 1 illustrates the conventional design flow of such systems. Starting from (i) a high-level specification of the controller-plant setup, (ii) control algorithms are typically designed based on well-defined semantics of the plant and the controller to satisfy specified *control goals* such as stability, settling time, or tracking error. Due to the distributed nature of processing resources, the control applications need to be (iii) partitioned into a number of software tasks for sampling the plant outputs, computing the control laws, and performing actuations. These software tasks need to be (iv) mapped and scheduled on different ECUs. Depending on the task mappings, control signals need to be exchanged via a shared, arbitrated network. Hence, (v) data processed by the tasks, e.g., feedback signals, are packetized as messages which need to be scheduled accordingly. As the control applications have to be designed to satisfy stringent quality and performance requirements, the actual control performance after implementation on the distributed platform might deviate from the desired behavior and violate the specified control goals. This is because many of the assump-

tions on periodicity, signal latencies and jitter, that are made at control design level, no longer hold after implementation of the system on the target platform. Closing the semantic gap between the high-level control models and their actual implementation on a distributed platform necessitates joint design methods. Such joint design under the constraints introduced by the platform-specific properties turns out to be a non-trivial task. For instance, the implementation platform imposes constraints on the available resources, e.g., bus identifiers and processing capacity, whereas the control behavior depends on the choice of scheduler parameters for both, tasks and messages, which are driven by the real-time and performance requirements of the control applications.

The FlexRay protocol has been developed to provide a robust, scalable, deterministic, and fault-tolerant communication system for advanced automotive control applications [2]. In particular, its high bandwidth of 10 Mbit/s, and the deterministic communication paradigm in the static segment supporting synchronization of sensor tasks, control functions and actuators, provide providential real-time properties for the implementation of distributed control systems [3]. However, the *configuration* of FlexRay systems is a complex task as the protocol provides more than 70 interdependent configurable parameters [4, 5]. The influence of real-time constraints on the design of FlexRay networks has been discussed in [6]. Further, quite some research effort has been spent on timing- and schedulabililty analysis of the FlexRay protocol. The work in [7] presents a timing analysis technique for the static and dynamic segment of the FlexRay protocol. The synthesis of communication schedules for the FlexRay static segment has been presented in [8, 9, 10]. In this context, [9] considers the synchronization of tasks and messages based on OSEK and OSEKtime real-time operating systems. The scheduling problem for the dynamic segment has been addressed in [11]. Further, there has been a significant amount of work in the area of control and scheduling co-design to improve the regulation of the interacting dynamics between the cyber and the physical parts along the lines of [12, 13, 14]. The work in [15, 16] discusses the synthesis of FlexRay schedules under consideration of stability and control performance objectives. However, to the best of our knowledge, none of the previous research efforts proposed an integrated design approach for the synthesis of controller parameters along with the scheduler parameters for all control-related tasks and messages.

1.1 Contributions and overview of our scheme

In this work, we present a constraint-driven design approach for the synthesis of controller parameters such as controller gains along with the control-related task and message schedules for a FlexRay-based implementation platform. In Section 2 we introduce feedback control models and the details of the implementation platform under consideration. Section 3 presents our proposed scheme as illustrated in Fig. 2. The overall objective in controller design is to satisfy specified *control goals* to guarantee the desired functionality of the system. Towards this, the *controller design* (stage 1) essentially boils down to the problem of finding suitable controller gains in order to achieve the desired control performance. The control performance depends on how *fresh* or old a feedback signal is (this is used to compute the control input). We will refer to such timing constraints as freshness constraints. For instance, the scheduling on the ECUs

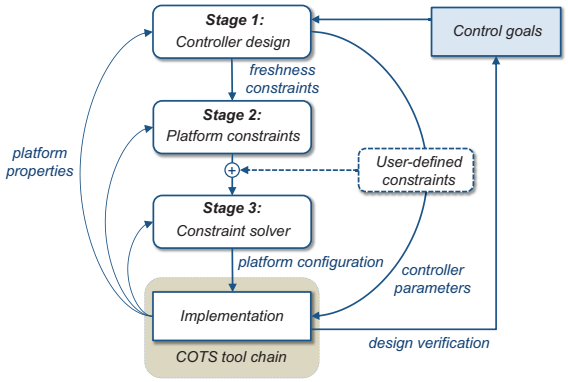

Figure 2: Constraint-driven design flow.

and on the FlexRay bus has an impact on the delays in the feedback signals which might deteriorate the control performance of the system. On the other hand, the available sampling intervals that can be realized by the implementation platform depend on the configuration of the FlexRay bus and the utilization of the processors on the ECUs. Towards this, we propose a joint design approach that (i) takes into account platform-specific properties for the *controller design*, and (ii) translates the freshness constraints on the feedback signals derived at controller design level into *platform constraints* (stage 2) that must be satisfied when actually implementing the control-related tasks and messages on the operating systems and the FlexRay bus.

FlexRay-based System Design: To obtain a feasible platform configuration satisfying the above constraints, we use a *constraint solver* (stage 3) which considers the platform constraints as well as optional *user-defined constraints*. In case no feasible platform parameters can be generated, we need to relax our control goals. Finally, the resulting platform- and controller parameters serve as an input for the design and *implementation* of the system using a COTS tool chain such as in [17, 18] which require both control laws and FlexRay parameters to be specified. Consequently, this technique significantly eases and complements the existing design process since these tools do not provide any mechanisms for automatically deriving the platform parameters from specified control performance constraints. The work we present here closes this gap and in addition enables control-architecture co-design. In Section 4, we show experimental results for the implementation of a DC motor control application and a car suspension system on a FlexRay HiL-setup using the synthesized controller- and platform parameters. We will also study how different bus configurations affect the controller design as well as the task and message schedules.

2. SYSTEM MODEL

2.1 Feedback control model

We start our analysis with a continuous-time system of the form

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t), \\ y(t) &= C_c x(t), \end{aligned} \tag{1}$$

where $x(t)$ is the $n \times 1$ vector for *state variables*, $u(t)$ is the *control input* to the system and $y(t)$ is the *output*. A_c is a $n \times n$ system matrix, B_c and C_c are input and output matrices of appropriate dimension. Subsequently, the continuous-time system is sampled at a constant sampling interval h. The

resultant system is a discrete-time system of the form

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k], \\ y[k] &= Cx[k], \end{aligned} \tag{2}$$

where

$$A = e^{A_c h}, B = \int_0^h (e^{A_c t} dt) \cdot B_c, C = C_c. \tag{3}$$

Such feedback control systems have two physical components: *actuators* (or plants) and *sensors*. We consider an architecture where the actuators and the sensors are spatially distributed and connected to different ECUs which communicate via a FlexRay bus. A feedback *controller* is an algorithm to compute $u[k]$ as a function of the states $x[k]$ or output $y[k]$ (feedback signals) such that $x[k]$ or $y[k]$ behave according to the specified control goals.

In this work, we assume all states $x[k]$ to be *measurable* and we use full-state feedback controllers of the form

$$u[k] = Kx[k], \tag{4}$$

where K are $1 \times n$ *state-feedback gains*. The controller design essentially boils down to choosing K such that the system (i) is *stable*, i.e., the closed-loop system matrix $(A+BK)$ has all the *eigenvalues* within the *unit circle*, and (ii) meets performance requirements in steady-state and transient phases, e.g., minimization of the tracking error and the settling time of the control system.

2.2 Implementation platform

Let us consider an automotive in-vehicle network where several ECUs are connected via a FlexRay bus as depicted in Fig. 3. The software implementation of distributed control applications running on such architectures is typically realized by model-based software development. This involves automatic code generation from high-level control models such as MATLAB/Simulink, along with the FlexRay driver stack and the operating system (OS). For this, the high-level control models are partitioned into several software tasks that need to be mapped on different ECUs. Subsequently, the generated code, e.g., C-code, is cross-compiled for the target hardware and flashed on the dedicated ECUs.

FlexRay ECU. A FlexRay ECU as depicted in Fig. 3 consists of (i) a host microcontroller running the OS which schedules and executes application tasks T_i and communication tasks T_{com}, (ii) a communication controller that implements the FlexRay protocol and (iii) bus drivers that realize the conversion of the logical bit stream into physical signals propagated on the bus. We consider a periodic time-driven non-preemptive scheduling policy implemented by the OS. It provides a task dispatcher, which allows cyclic task execution, synchronous and asynchronous to the FlexRay schedule. Let a dispatch event for a task T_i (see Fig. 4) be defined by the tuple $T_i = \{o_i, p_i, e_i\}$:

- The task offset o_i specifies the duration from start time to the first task invocation of T_i.
- The task period p_i specifies the time between two consecutive task activations of T_i.
- The worst-case execution time e_i specifies the time it takes to execute T_i in the worst case.

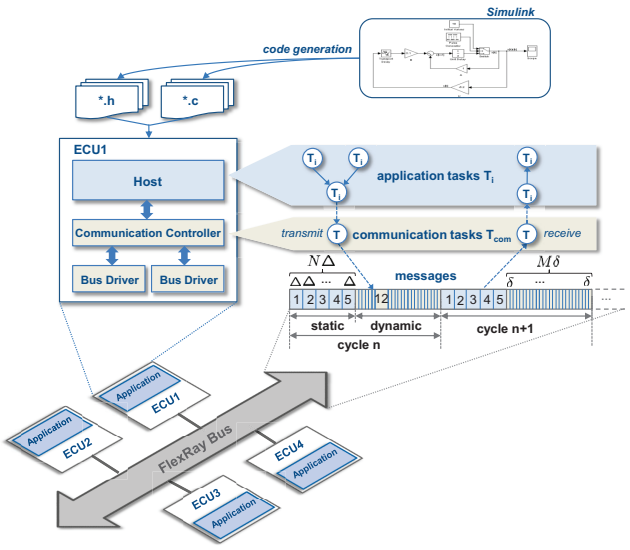

Figure 3: FlexRay implementation platform.

Hence, the k-th instance of task T_i is triggered at

$$t_i^k = o_i + kp_i, \ k \in \mathbb{N}_0. \tag{5}$$

The scheduler of the OS processes all tasks according to a dispatch table in a cyclic manner where the length of the dispatch table is determined by the *hyperperiod* $H = \underset{i}{lcm}(p_i)$. Consequently, all task invocation times can be computed offline and stored in the dispatch table. Further, the k-th instance of T_i finishes at the latest

$$\tilde{t}_i^k = o_i + kp_i + e_i, \ k \in \mathbb{N}_0. \tag{6}$$

Fig. 4 shows an example where five tasks T_i, $i \in \{1, ..., 5\}$, are mapped on three different ECUs. The computed outputs of T_1, T_2, and T_4 are packetized as messages m_1, m_2, and m_4 which are transmitted on the bus according to their specified FlexRay schedules. For this, communication tasks T_{com} write the output of the corresponding application tasks to the dedicated transmit buffers of the communication controller as depicted in the figure. Similarly, in case a message is received by an ECU, a communication task reads the corresponding receive buffer and forwards the unpacked data to the application task for further processing, e.g., m_1 gets processed by T_3 in Fig. 4. Hence, every communication task generates dispatch events according to (5) and needs to be considered in the dispatch tables. In this work we are not interested in explicitly synthesizing dispatch events for the communication tasks. Instead, we account for sufficiently large time windows of length ϵ that capture the communication stack overhead such as frame packing and unpacking times and write/read operations to/from the buffers of the communication controllers. Consequently, during the window ϵ communication tasks can easily be configured either manually or automatically using tool support as in [17, 18].

FlexRay bus. The FlexRay communication is organized as a sequence of 64 bus cycles that are periodically repeated [2]. The cycles are indexed by a cycle counter from 0 to 63 after which the counter is reset to 0 again. Each cycle is of fixed duration T_{bus} and consists of a static and a dynamic segment. Further, each cycle finishes with a short Network Idle Time (NIT) segment during which the clock synchro-

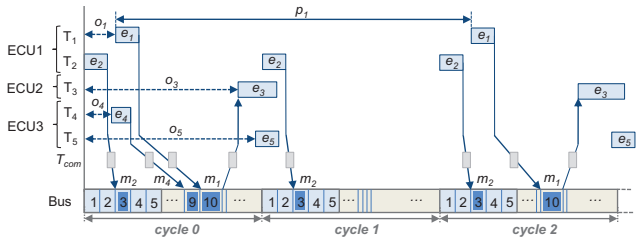

Figure 4: Scheduling of tasks and messages.

nization is performed and no bus communication is possible. Messages m_i are transmitted on the FlexRay bus in either static or dynamic segments according to their predefined bus schedules. We refer to a bus schedule as the tuple $\Theta_i = \{S_i, B_i, R_i\}$.

The communication slot S_i determines the time window in which a message may be transmitted. Each static segment consists of N slots, denoted by $S_i \in \{1, ..., N\}$, that are of fixed and equal length Δ as depicted in Fig. 3, e.g., $N = 5$. For instance in Fig. 4, m_2 is assigned $S_2 = 3$ in every cycle of the static segment. The dynamic segment is partitioned into M equal-length *minislots* of fixed duration $\delta << \Delta$. If a message is assigned a slot in the dynamic segment, i.e., $S_i \in \{N+1, ..., N+M\}$, it takes several minislots to transmit m_i depending on its size. In case no message is transmitted, only one minislot is consumed by the bus and the slot number is incremented with the next minislot. This is illustrated in Fig. 4, where m_1 is assigned slot $S_1 = 10$ in the dynamic segment in every even cycle. In *cycle 1*, m_1 is not transmitted and hence only one minislot is consumed. Note that compared to *cycle 2*, m_1 gets delayed in *cycle 0* because of the transmission of m_4 which has a higher priority than m_1. Hence, messages in the dynamic segment may experience variable delays in every cycle depending on the interference due to messages with higher priorities. Further, a message can only get transmitted in the dynamic segment if its slot is available in the current cycle, i.e., there are enough minislots available to transmit m_i in S_i. The last minislot a message is allowed to start a transmission is denoted by the parameter $pLatestTx$. The base cycle $B_i \in \{0, 1, ..., 63\}$ specifies the first cycle during which S_i is available. For instance in Fig. 4, m_2 is assigned $B_2 = 0$ as the first cycle in which $S_2 = 3$ is available is *cycle 0*. The repetition rate $R_i \in \{1, 2, 4, 8, 16, 32, 64\}$ determines the number of cycles that must elapse between two feasible transmissions. For example, m_1 is assigned repetition rate $R_1 = 2$, and hence in every second cycle S_1 is available. Similarly, m_2 is assigned $R_2 = 1$, therefore S_2 is available in every cycle. Further, $B_i < R_i$ holds. According to the above definitions, the schedules for m_1 and m_2 denoted in Fig. 4 are $\Theta_1 = \{10, 0, 2\}$, and $\Theta_2 = \{3, 0, 1\}$.

Platform configuration. For each control application the set of n control-related tasks is denoted by T_i, $i \in \{1, ..., n\}$ and schedules for m messages triggered T_i is denoted by Θ_i. Thus, a control application $\mathcal{C}_l$ has a $(n + m)$-tuple of platform parameters $\Phi_l = \{T_i, \Theta_i\}$. We also call Φ_l a valid *platform configuration* for $\mathcal{C}_l$. Note that a message that is triggered by task T_i is assigned a schedule $\Theta_i = \{S_i, B_i, R_i\}$, otherwise $\Theta_i = \{\}$. Let us consider a control application $\mathcal{C}_l$ that is partitioned into $n = 5$ tasks that trigger $m = 3$ messages, as depicted in Fig. 4. Then the corresponding platform parameters are denoted by the 8-tuple $\Phi_l = \{T_1, T_2, T_3, T_4, T_5, \Theta_1, \Theta_2, \Theta_4\}$. Recall that $T_i = \{o_i, p_i, e_i\}$, and $\Theta_i = \{S_i, B_i, R_i\}$ are tuples itself. As only T_1, T_2, and T_4 trigger messages on the bus, $\Theta_3 = \{\}$, and $\Theta_5 = \{\}$. The goal is to synthesize a valid platform configuration Φ_l that satisfies all control performance requirements of $\mathcal{C}_l$.

3. CONSTRAINT-DRIVEN CO-DESIGN

In the following we outline our proposed co-design approach in three stages. In the first stage, we design the control law while taking into account platform-specific properties and a freshness constraint for the feedback signals. In stage 2, we translate the specified freshness constraint into several platform constraints by considering all implementation-specific platform details. Finally, in stage 3, we synthesize a platform configuration $\Phi_l \in \{T_i, \Theta_i\}$, i.e., all control-related task and message schedules, that satisfy all the constraints.

3.1 Stage 1: Controller design

In this stage, we will discuss how platform-specific properties are already considered during controller design to optimize control performance and to derive constraints for the platform configuration $\mathcal{C}_l$. In particular, we derive the controller gains K for $\mathcal{C}_l$, the periods $p_i \in T_i$ for all control-related tasks, and the repetition rates $R_i \in \Theta_i$, for which the desired control goals are satisfied. For the implementation of the discrete-time control system model shown in (2), the sampling period needs to be a constant h. Therefore, all control-related tasks T_i belonging to $\mathcal{C}_l$ have $p_i = h$. Towards this, we require the sampling period h to be harmonic with respect to the FlexRay bus cycle T_{bus}, i.e., h is a multiple of the feasible bus periods $h = R_i T_{bus}$. Depending on R_i, the bandwidth utilization of the bus changes, i.e., the smaller the value of R_i, the more are the number of cycles available to transmit a message m_i, and in turn less cycles are available for future messages. At the same time, the sampling time h also changes based on the choice of R_i. Hence, the performance of the control application changes with h and R_i. Consequently, we are interested in computing the optimal value of the sampling period h while maximizing R_i in order to increase the bandwidth utilization of the bus. This yields

$$h = \bar{R} T_{bus}, \tag{7}$$

where $\bar{R}$ indicates the highest $R_i \in \Theta_i$ among all control-related schedules Θ_i that belong to $\mathcal{C}_l$. In the state-feedback controller in (4), the control input $u[k]$ utilizes the values of the states $x[k]$ at the k-th sampling instant. Therefore, the feedback is assumed to be applied instantaneously without any delay τ, i.e., the time difference between reading the sensor values and applying the actuation is equal to zero. We denote τ as the sensor-to-actuator delay. However, in a distributed platform, clearly task processing and message transmissions consume a significant amount of time, i.e., $\tau >> 0$, before the control input can be applied to the actuator. Hence, neglecting τ when designing the control algorithm may lead to undesired behavior of the control system when actually implemented on the target platform.

In this work, we design the controller gains K of $\mathcal{C}_l$ using a *pole placement* technique [19] assuming constant sensor-to-actuator delays τ to achieve asymptotic stability of the system (2). To that effect, we ensure temporal determinism

of the feedback signals, i.e., realizing a constant delay of τ, by choosing an appropriate platform configuration Φ_l. As a result, we achieve a deterministic behavior of the controller $\mathcal{C}_l$ when actually implemented. More specifically, we intend to ensure that the feedback signals experience exactly one sampling interval delay and design the control law as

$$u[k] = Kx[k-1]. \tag{8}$$

Hence, the system will be asymptotically stable if all the feedback signals $x[k]$, $\forall k$ are delayed by $\tau = h$. Such design consideration poses restrictions on the freshness of the feedback signals.

Further, the dependency of control performance on the sampling interval h has been utilized to determine the optimal choice of sampling period [14]. Next, our goal is to find a feasible sampling period $h = \bar{R}T_{bus}$ that minimizes specified cost functions. Here, we consider two cost functions. The steady-state performance of a control application is measured by the commonly used cost function

$$J_1 = \sum_{k=0}^{N} \int_{kh}^{(k+1)h} [\lambda u(t)^2 + (1-\lambda)e(t)^T e(t)]dt, \tag{9}$$

where λ is a weight, $u(t)$ is the control input and $e(t) = |y(t) - r|$ is the error between the reference value r and the output $y(t)$. The first term in (9) accounts for the energy input into the system where the second term quantifies the tracking error. According to (7) only discrete sampling periods h are available depending on the bus cycle length T_{bus} and the available repetition rates R_i. As T_{bus} is a global network parameter that is already fixed at the design time of the network, we are interested in finding the repetition rate $\bar{R}$ for all control-related messages that realize

$$J_1^* = \min_{\bar{R}} \left(\sum_{k=0}^{N} \int_{k\bar{R}T_{bus}}^{(k+1)\bar{R}T_{bus}} [\lambda u(t)^2 + (1-\lambda)e(t)^T e(t)]dt \right). \tag{10}$$

The transient performance of control systems is measured by its *settling* time ξ. We determine the *settling* time ξ by simulation according to the amount of time that the control system requires to reach and remain within 1% of the reference value r

$$J_2^* = \min_{\bar{R}} \xi. \tag{11}$$

Based on the characteristics of the control applications it is decided which cost function needs to be minimized. Alg. 1 illustrates the control design flow of this stage. We start with a continuous-time system (line 1). Next, we discretize the continuous-time system at the feasible sampling periods h (line $3-5$), depending on the choice of T_{bus} and repetition rates. Subsequently, we compute the controller gains K for the corresponding discrete-time system (2) using pole placement and simulate the system for a sufficiently long time, i.e., N samples, with $u[k]$ being as per (8) (line $8-10$). For each run, we compute the transient and/or steady-state performance (line 11). Finally, we select the repetition rate $\bar{R}$ and sampling period h for which the selected cost functions (10), (11) were minimized (line 13).

3.2 Stage 2: Platform constraints

In the previous stage we outlined the control algorithm design, taking into account a constant sensor-to-actuator delay for the feedback signals. Next we translate the freshness constraint into platform constraints that realize (i) synchronicity, (ii) signal correlation, and (iii) schedulability on the implementation platform such that $\tau = h$.

Algorithm 1 Controller design under platform constraints.

Require: T_{bus}, λ, $A_c, B_c, C_c, x(0)$
1: contSystem(A_c, B_c, C_c) //continuous-time system (1)
2: **for all** $i \in \{0, ..., 6\}$ **do**
3: $\bar{R} = 2^i$
4: $h = T_{bus} \cdot \bar{R}$ //compute feasible sampling period
5: c2d(A_c, B_c, C_c, h) //conversion to discrete-time system
6: K = computeGains() //pole placement
7: u = computeControlLaw() //equation (8)
8: **while** simulate($x[0], N$) **do**
9: getResponse() //for N samples using (2) and (8)
10: **end while**
11: J_i = getPerformance() //equation (9) or ξ
12: **end for**
13: $J_i^* = \min_i J$

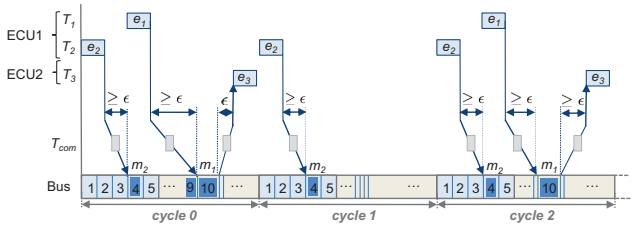

Figure 5: Synchronicity and schedulability considering communication stack overhead.

Synchronicity constraint. The output data of T_i is packetized as a message m_i and transmitted over the bus according to Θ_i. The synchronicity constraint specifies the phase requirement between the task finish time $\tilde{t}_i$ of T_i and the corresponding bus schedule Θ_i. The message m_i can be transmitted either via static segment or dynamic segment as per Θ_i. Let T_i release a message m_i with schedule Θ_i, then

$$\tilde{t}_i^k + \epsilon < t(\Theta_i), \ \forall k \in \mathbb{N}_0 \tag{12}$$

where $\tilde{t}_i^k$ is the finish time of task T_i, and $t(\Theta_i)$ denotes the starting time of slot S_i according to $\Theta_i = \{S_i, B_i, R_i\}$ as

$$t(\Theta_i) = B_i T_{bus} + k\bar{R}T_{bus} + (S_i - 1)\Delta, \ \forall k \in \mathbb{N}_0. \tag{13}$$

The first term $B_i T_{bus}$ in (13) captures the cycle offset, i.e., the starting time of the first cycle where S_i is available at $k = 0$. The second term $k\bar{R}T_{bus}$ denotes the relative cycle offset depending on the sampling instant k where S_i is available, and $(S_i - 1)\Delta$ accounts for the slot offset within a cycle until S_i is available. The value of ϵ in (12) accounts for a sufficiently large time interval that captures the FlexRay driver overhead during which the communication task T_{com} can be configured to write the output data of T_i to the FlexRay controller before S_i is available (see Fig. 5). Using (6), (7), and (13) in (12) yields

$$o_i + e_i + \epsilon < B_i T_{bus} + (S_i - 1)\Delta. \tag{14}$$

Note that (14) only depends on the bus schedule parameters $S_i, B_i \in \Theta_i$ and the task offset o_i, and is independent of the sampling instance k.

If m_i is scheduled in the dynamic segment, $t(\Theta_i)$ defines the earliest point in time S_i is available according to

$$t(\Theta_i) = B_i T_{bus} + k\bar{R}T_{bus} + N\Delta + (S_i - 1 - N)\delta, \ \forall k \in \mathbb{N}_0 \tag{15}$$

In (15), $N\Delta$ captures the blocking time of the static segment, and $(S_i - 1 - N)$ captures the number of minislots that elapse until S_i is available in the best case, i.e., no message is transmitted with higher priority than m_i. Using (6), (7), and (15) in (12) yields

$$o_i + e_i + \epsilon < B_i T_{bus} + N\Delta + (S_i - 1 - N)\delta, \qquad (16)$$

Fig. 5 illustrates an example with two tasks $T_1 = \{o_1, p_1, e_1\}$, $T_2 = \{o_2, p_2, e_2\}$ releasing m_1 in the dynamic segment with $\Theta_1 = \{10, 0, 2\}$, and m_2 in the static segment with $\Theta_2 = \{4, 0, 1\}$. The figure shows, that in the dynamic segment the actual starting time of a slot is variable and depends on the interference of messages having higher priorities. For instance, in *cycle 2*, slot $S_1 = 10$ starts earlier compared to *cycle 0* where S_1 gets delayed due to a message transmitted in slot 9.

Correlation constraint. The freshness constraint $\tau = h$ implies that the control input $u[k]$ is utilized for actuation at sampling instant $(k+1)$. The control law in (8) requires the control input $u[k]$ to be computed based on the state $x[k]$ at sampling instant k to meet the freshness constraint. Towards this, the correlation constraint specifies the maximum allowed time skew among input signal $u[k]$ and each element of the measured state $x[k]$ that is used to compute the input. In the following, let us consider the example illustrated in Fig. 6 where a control application $\mathcal{C}_1$ is partitioned into four tasks and three messages. Let the tasks and messages be denoted with additional subscripts s, c, a for sensor-, controller-, and actuator-related data. In the example, $s \in \{1, 2\}$, $c \in \{1\}$, and $a \in \{1\}$. The sensor tasks $T_{1,1}$, and $T_{2,2}$ read the states of the continuous signals $x_1(t)$, and $x_2(t)$ at every sampling interval h. The sampled data is sent to the controller task $T_{3,1}$ that computes the control law. The computed control input $u[k]$ is sent to the actuator which applies the control input to the system using $T_{4,1}$. It is clear from the figure that the freshness constraint on τ is violated at sampling instance $k = 4$ as the control input $u[4]$ gets delayed by more than one sampling interval, and hence arrives at the actuator task $T_{4,1}$ not before $k = 5$. Hence, $u[4]$ will finally be applied at $k = 6$ which results in $\tau = 2h$ or $u[4]$ will be overwritten by $u[5]$.

We require (i) the time skew between all measured outputs $x[k]$ to be zero, i.e., all sensor tasks $T_{i,s}$ of $\mathcal{C}_l$ are triggered at the same time instances $t^k_{1,1} = t^k_{2,2}$ (see sensor tasks $T_{1,1}$, $T_{2,2}$ in Fig. 6), and (ii) the control input $u[k]$ computed by $T_{i,c}$ to be applied at the actuator task $T_{i,a}$ at $\tau = h$ (see $T_{3,1}$, and $T_{4,1}$ in Fig. 6). This realizes the freshness constraint according to the control law defined in (8). As the task invocations of sensor and actuator tasks coincide with each sampling instance we have $t^{k+1}_{i,a} - t^k_{i,s} = h$ which yields

$$o_{i,a} + (k+1)h - (o_{i,s} + kh) = h, \ \forall k \in \mathbb{N}_0, \qquad (17)$$

and finally

$$o_{i,a} = o_{i,s}, \qquad (18)$$

Consequently, we design the schedules of sensor tasks $T_{i,s}$ and actuator tasks $T_{i,a}$ with equal task offsets, i.e., control input is applied at equidistant time intervals and delayed by one sampling interval h. In other words, (18) imposes constraints on the platform configuration to realize the freshness assumption $\tau = h$ that has been made in stage 1 (see (8)) to design the control law. Provided that all synchronicity

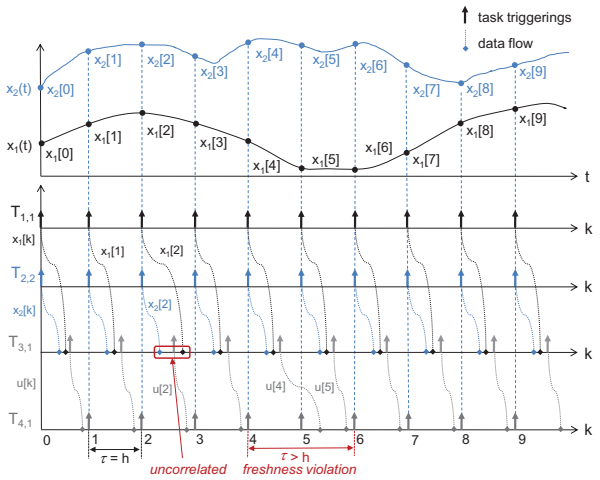

Figure 6: Correlation constraint.

constraints are fulfilled for T_i and Θ_i, each control-related message m_i carries the up-to-date data of its corresponding sender task, i.e., no message misses its slot, and hence the most recent data is transmitted over the bus. Further, the earliest receiving of a sensor value that is transmitted in slot $S_{i,s}$ before executing $T_{i,c}$ is bounded by

$$o_{i,c} > \max_{\forall s}(B_{i,s} T_{bus} + S_{i,s}\Delta) + \epsilon. \qquad (19)$$

For instance, in Fig. 6 at sampling instant $k = 2$ the value of $x_1[2]$ arrives at the controller after the controller task $T_{3,1}$ has been triggered at $k = 2$, and hence the control law is computed based on $x_1[1]$ and $x_2[2]$ which may lead to an unpredictable control input $u[2]$.

The actuator task $T_{i,a}$ is triggered not before the control input has arrived according to $\Theta_{i,c} = \{S_{i,c}, B_{i,c}, \bar{R}\}$

$$o_{i,a} + h > B_{i,c} T_{bus} + S_{i,c}\Delta + \epsilon. \qquad (20)$$

Similarly, in case the control-related messages are sent via the dynamic segment of FlexRay, the conditions (19) and (20) become

$$o_{i,c} > \max_{\forall s}\big(B_{i,s} T_{bus} + (n_{i,s} + c_{i,s})\delta\big) + N\Delta + \epsilon \qquad (21)$$

and

$$o_{i,a} + h > B_{i,c} T_{bus} + (n_{i,c} + c_{i,c})\delta + N\Delta + \epsilon. \qquad (22)$$

where the worst-case minislot consumption is bounded by

$$n_i = \sum_{j \in hp_i} (c_j - 1) + (S_i - N - 1) < pLatestTx \qquad (23)$$

In (23), c_j denotes the message size in terms of minislots. The first term $\sum_{j \in hp_i}(c_j - 1)$ captures the worst-case interference due to messages m_j having a higher priority than m_i and the second term in (23) accounts for the number of minislots that elapse until S_i is available. Note that the value of $pLatestTx$ denotes the last minislot a message transmission is allowed to start in the dynamic segment.

Schedulability constraint. A feasible platform configuration Φ_l needs to satisfy the schedulability constraints (i) of all tasks $T_i \in \Phi_l$ that are executed on the ECUs, and (ii) all bus schedules $\Theta_i \in \Phi_l$ of the control-related messages. Let $T_j \in \mathcal{T}_N$ be the set of existing tasks that is mapped on ECU N. Further, let all periods be harmonic and multiples of the

FlexRay communication cycle length T_{bus}. Then, a feasible dispatch table is realized if there exists no overlap between any task executions within a *hyperperiod* $H_N = lcm(p_j, p_i)$. We obtain

$$(\tilde{w}_i^k < w_j^{k'}) \vee (w_i^k > \tilde{w}_j^{k'}),\ \forall i,j,\ k \in \frac{H_N}{p_i}, k' \in \frac{H_N}{p_j}. \quad (24)$$

where

$$\tilde{w}_i^k = \begin{cases} \tilde{t}_i^k + \epsilon, & \text{if } \textit{task } T_i \textit{ triggers a message} \\ \tilde{t}_i^k, & \textit{otherwise} \end{cases} \quad (25)$$

and

$$w_i^k = \begin{cases} t_i^k - \epsilon, & \text{if } \textit{task } T_i \textit{ receives a message} \\ t_i^k, & \textit{otherwise} \end{cases} \quad (26)$$

For any application tasks triggering or receiving messages, we consider a sufficiently large time window ϵ that captures the communication task overhead due to T_{com} as illustrated in Fig. 5. Consequently, for tasks triggering messages, the first condition in (24) becomes $\tilde{w}_i^k = \tilde{t}_i^k + \epsilon$ according to (25) as in the case of T_1, and T_2 in Fig. 5. Similarly, for any task that processes data of a receiving message, e.g., T_3 processing m_1, we have $w_i^k = t_i^k - \epsilon$ according to (26).

Further, let $\Theta_j \in \Omega$ be the set of schedules that is mapped on the FlexRay bus. Then, $\Theta_i \in \mathcal{C}_l$ are feasible FlexRay schedules according to

$$(S_i = S_j) \rightarrow (B_i + n \cdot R_i) \neq (B_j + n' \cdot R_j) \quad (27)$$
$$\forall i,j,\ n \in \{0, ..., \frac{64}{R_i} - 1\}, n' \in \{0, ..., \frac{64}{R_j} - 1\}$$

i.e., any intersection between bus cycles of schedules sharing the same slot is prohibited. Further, in the static segment, every ECU N is owner of a slot S_i, i.e. messages transmitted by different ECUs may not share the same slot. In the dynamic segment, different ECUs may share a slot in different cycles using *cycle multiplexing*.

3.3 Stage 3: Platform configuration synthesis

The platform constraints derived in the previous stage are used to formulate a Constraint Satisfaction Problem (CSP) to synthesize a feasible platform configuration Φ_l for any control application $\mathcal{C}_l$. Recall that we already derived $R_i = \bar{R}$, and $p_i = h$ during the controller design in stage 1. We would like to mention that we do not address the problem of worst-case execution time analysis within the scope of this work, and hence assume the worst-case execution time e_i of all tasks to be given. We define a CSP as a set of the variables $o_i \in T_i$, and $S_i, B_i \in \Theta_i$ with finite domains

$$\forall\ 0 \leq o_i < h - e_i, \quad (28)$$

$$\forall\ a \leq S_i \leq b, \quad (29)$$

where $a = 1, b = N$ in case S_i is a static slot, and $a = N+1, b = N+M$ in case S_i is a dynamic slot.

$$\forall\ 0 \leq B_i < \bar{R}, \quad (30)$$

Next, we apply the set of constraints derived in stage 2 which define the relations that must hold among the values of the variables, i.e., (i) synchronicity (14), (16), (ii) correlation as defined in (19), (20), (21), (22), and (iii) schedulability on ECUs (24), and the FlexRay bus (27). Finally, we obtain a valid platform configuration $\Phi_l = \{T_i, \Theta_i\}$ that realizes the schedules of all control-related tasks and messages while satisfying the imposed constraints. Note that user-defined constraints may be introduced to either restrict the finite domains, e.g., select specific slot ranges for scheduling the applications, or impose other non-functional constraints.

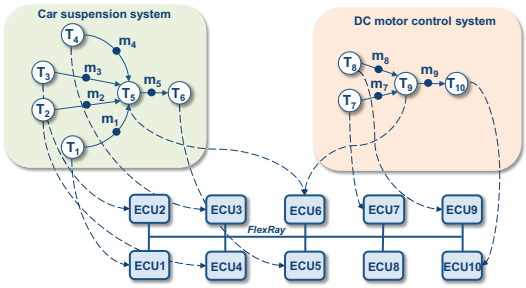

Figure 7: Task partitioning and mapping.

4. EXPERIMENTAL RESULTS

We show the effectiveness of our approach using two common automotive control applications: (i) a DC motor speed control application $\mathcal{C}_{DC}$, and (ii) a car suspension system $\mathcal{C}_{CS}$. The continuous-time state-space representation of $\mathcal{C}_{DC}$ is adapted from the available model in [20]

$$A_{DC} = \begin{bmatrix} \frac{-\kappa_2}{\kappa_1} & \frac{\kappa_3}{\kappa_1} \\ \frac{-\kappa_3}{\kappa_5} & \frac{-\kappa_4}{\kappa_5} \end{bmatrix}, B_{DC} = \begin{bmatrix} 0 \\ \frac{1}{\kappa_5} \end{bmatrix}, C_{DC} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad (31)$$

where $\kappa_1 = 0.01\ kgm^2/s^2$, $\kappa_2 = 0.1\ Nms$, $\kappa_3 = 0.01\ NmA$, $\kappa_4 = 0.75\ \Omega$, and $\kappa_5 = 0.05\ H$. The state variables $x = [\ x_1 \quad x_2\ ]^T$ are the rotational speed of the motor shaft and motor armature current, u is the motor terminal voltage.

Further, the car suspension system model $\mathcal{C}_{CS}$ is adapted from [21]

$$A_{CS} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-\sigma_3}{\sigma_1} & \frac{-\sigma_2}{\sigma_1} & \frac{\sigma_3}{\sigma_1} & \frac{\sigma_2}{\sigma_1} \\ 0 & 0 & 0 & 1 \\ \frac{\sigma_3}{\sigma_4} & \frac{\sigma_2}{\sigma_4} & -\frac{(\sigma_3+\sigma_6)}{\sigma_4} & -\frac{(\sigma_2+\sigma_5)}{\sigma_4} \end{bmatrix}, \quad (32)$$

$$B_{CS} = \begin{bmatrix} 0 \\ \frac{\sigma_2 \cdot \sigma_5}{\sigma_1 / \sigma_4} \\ \frac{\sigma_5}{\sigma_4} \\ \frac{(\sigma_6 - (\sigma_2+\sigma_5) \cdot \frac{\sigma_5}{\sigma_4})}{\sigma_4} \end{bmatrix}, C_{CS} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (33)$$

where $\sigma_1 = 100\ kg$, $\sigma_2 = 400\ Ns/m$, $\sigma_3 = 800\ N/m$, $\sigma_4 = 10\ kg$, and $\sigma_5 = 200\ Ns/m$, and $\sigma_6 = 800\ N/m$. The state variables x_1 and x_2 are the positions and velocities of the body of the car and x_3 and x_4 are the positions and velocities of the mass of the suspension system. The input u is the force applied to the body by the suspension system. We consider the task partitioning and mappings of both applications to be given according to Fig. 7. Here, $\mathcal{C}_{CS}$ consists of $n = 6$ tasks where each of the four sensor tasks T_1, T_2, T_3, and T_4 triggers a message m_i on the FlexRay bus that needs to be scheduled according to $\Theta_1, \Theta_2, \Theta_3$, and Θ_4. The control law is computed by the controller task T_5 which sends the control input to the actuator task T_6 via message m_5.

Further, $\mathcal{C}_{DC}$ consists of $n = 4$ tasks where the two sensor tasks T_7, T_8 trigger m_7, m_8 according to Θ_7, Θ_8, respectively. The controller task T_9 computes the control input which is transmitted via m_9 corresponding to Θ_9 to the actuator task T_{10}. Note that the controller tasks of both appli-

Table 1: FlexRay bus configuration parameters.

Parameters	Config. I	Config. II
bus speed [Mbit/s]	10	10
cycle length $T_{bus}[ms]$	5	12
#static slots N	25	40
static slot length $\Delta[ms]$	0.10	0.16
#minislots M	230	540
minislot length $\delta[ms]$	0.01	0.01
pLatestTx	225	535

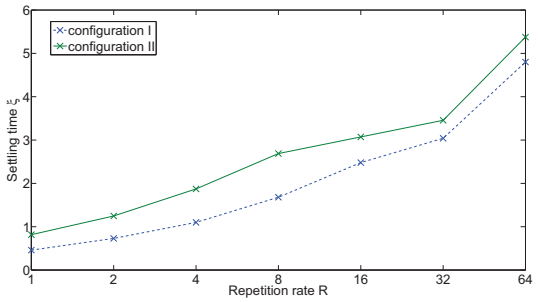

Figure 8: Transient performance of $\mathcal{C}_{CS}$ for configuration I and II.

cations share the same processing resources on ECU6. Also, we assume that each ECU is synchronized with the FlexRay bus and executes a synchronization task T_{sync} which takes care of the synchronization of the local ECU clocks with the global time base on the bus. Further, we consider ECU8 to transmit 15 non-control messages on the bus, where 5 periodic messages are sent in the static segment and 10 event-triggered messages are transmitted in the dynamic segment. Our goal is to synthesize the controller gains K_{CS}, K_{DC}, and the platform configurations Φ_{CS} and Φ_{DC} while satisfying all control objectives. Note that Φ_{CS} is an 11-tuple and Φ_{DC} is an 7-tuple. In the following we will consider two different bus configurations as depicted in Table 1. Towards this, we will study how different bus configurations affect the design of the controller gains K_{CS}, K_{DC}, and platform configurations Φ_{CS}, Φ_{DC}.

4.1 Stage 1: Controller design

At this stage, we design the controllers $\mathcal{C}_{CS}$, and $\mathcal{C}_{DC}$ in MATLAB/Simulink according to Alg. 1 presented in Section 3. Since the most important requirement of the car suspension system is a fast response time to road disturbances and vibrations we are mainly interested in the transient performance of $\mathcal{C}_{CS}$, and hence we choose (11) as the cost function to minimize the settling time ξ. The results for controller gains K_{CS}, sampling period h_{CS}, and repetition rate $\bar{R}_{CS}$ are depicted in Table 2 for both configurations. It can be seen in Fig. 8 that the minimum cost J_2^*, i.e., the smallest settling time ξ, is achieved at $\bar{R} = 1$, and therefore $h_{CS} = T_{bus}$ in both configurations. This is expected because faster sampling and actuation of the systems results in a better control performance of the system. For the DC motor application $\mathcal{C}_{DC}$ we require the motor to rotate the desired speed, i.e., the deviation from the reference value needs to be minimized while minimizing the required voltage of the control input to avoid damage of the motor. To account for both objectives we evaluate J_1^* using (10), and $\lambda_{DC} = 0.001$. Fig. 9 illustrates the quadratic cost J_1 over repetition rate $\bar{R}$ for both configurations. It is interesting to observe that in

Table 2: Results of stage 1: controller gain K, sampling period h and repetition rate $\bar{R}$.

Design parameters	Configuration I	Configuration II
$h_{CS}[ms]$	5	12
$\bar{R}_{CS}$	1	1
K_{CS}	[7712 2175 -12141 -61]	[594.9 189.6 -1001.5 -4.3]
$h_{DC}[ms]$	20	24
$\bar{R}_{DC}$	4	2
K_{DC}	[-4.7197 -0.5444]	[-3.2687 -0.4218]

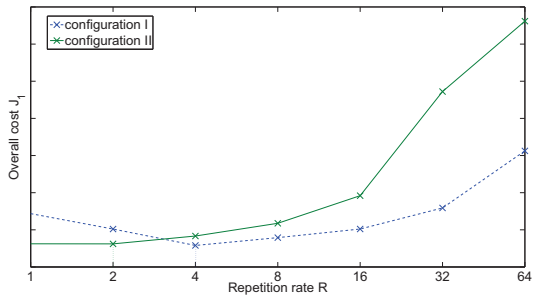

Figure 9: Steady-state performance of $\mathcal{C}_{DC}$ for configuration I and II.

configuration I the available repetition rate at the minimum cost J_1^* is $\bar{R}_{DC} = 4$ and $h_{DC} = 20ms$, whereas in *configuration II*, J_1 is minimum at $\bar{R}_{DC} = 2$, $h_{DC} = 24ms$ respectively. This is because the available sampling frequency h_{DC} that can be realized depends on the bus cycle lengths T_{bus} which is $5ms$ in *configuration I* and $12ms$ in *configuration 2*. Hence, the two configurations require different values of $\bar{R}$ to realize the optimal available sampling frequency. Note that the optimal sampling frequencies $h_{DC} = 20ms$, and $h_{DC} = 24ms$ in both configurations are quite close to each other, however they are realized by different repetition rates $\bar{R}_{DC}$. Results for K_{DC}, h_{DC}, and $\bar{R}_{DC}$ are shown in Table 2.

4.2 Stages 2 and 3: Platform configuration synthesis

Next, we utilize the results derived in stage 1 to formulate the CSP and generate feasible platform configurations for both the applications. We schedule the two control applications in a rate-monotonic fashion, i.e., we assign the highest priority π to the application with the smallest sampling period. Hence, $\pi_{CS} = 1$, and $\pi_{DC} = 2$ in both configurations, where $\pi_{CS} = 1$ denotes the highest priority. We choose $\epsilon = 3\Delta$ to consider the frame packing/unpacking times and buffer read/write operations of the communication tasks. The value of ϵ has been determined empirically, and hence might be conservative. Similarly, the worst-case execution times have been determined experimentally and considered with $e_i = 0.1ms$ for any control-related task of $\mathcal{C}_{CS}$ and $\mathcal{C}_{DC}$. The finite domains for the offsets o_i have been discretized in $0.1ms$ step size, i.e., $o_i \in \{0, 0.1, 0.2, ...\}$. We implemented the CSP for $\mathcal{C}_{CS}$ and $\mathcal{C}_{DC}$ in Python using the *Labix python-constraint library* [22] which is a Python module offering backtracking solvers, recursive backtracking solvers, and minimum conflicts solvers over finite domains. The CSP-solver has been executed on a dual core 1.8 GHz processor with 3 GB RAM. Table 3 shows average run-times of the different solvers. In every case a feasible solution could be determined in approximately 1 *sec* using an appropriate

Table 3: Average run-times of different CSP-solvers.

$\mathcal{C}_{CS}$	Backtracking	Rec. Backtracking	Min. Conflict
I	320 sec	**0.2 sec**	1.5 sec
II	5384 sec	**1.3 sec**	81.4 sec
$\mathcal{C}_{DC}$	Backtracking	Rec. Backtracking	Min. Conflict
I	546 sec	8.7 sec	**0.6 sec**
II	9295 sec	3900 sec	**0.1 sec**

Table 4: Φ_{CS} and Φ_{DC} for static segment.

	Configuration I		Configuration II	
i	T_i of $\mathcal{C}_{CS}$	Θ_i of $\mathcal{C}_{CS}$	T_i of $\mathcal{C}_{CS}$	Θ_i of $\mathcal{C}_{CS}$
1	$\{0.5, 5, 0.1\}$	$\{11, 0, 1\}$	$\{3.5, 12, 0.1\}$	$\{27, 0, 1\}$
2	$\{0.5, 5, 0.1\}$	$\{12, 0, 1\}$	$\{3.5, 12, 0.1\}$	$\{28, 0, 1\}$
3	$\{0.5, 5, 0.1\}$	$\{13, 0, 1\}$	$\{3.5, 12, 0.1\}$	$\{29, 0, 1\}$
4	$\{0.5, 5, 0.1\}$	$\{14, 0, 1\}$	$\{3.5, 12, 0.1\}$	$\{30, 0, 1\}$
5	$\{1.8, 5, 0.1\}$	$\{24, 0, 1\}$	$\{5.3, 12, 0.1\}$	$\{38, 0, 1\}$
6	$\{0.5, 5, 0.1\}$	-	$\{3.5, 12, 0.1\}$	-
i	T_i of $\mathcal{C}_{DC}$	Θ_i of $\mathcal{C}_{DC}$	T_i of $\mathcal{C}_{DC}$	Θ_i of $\mathcal{C}_{DC}$
7	$\{10.2, 20, 0.1\}$	$\{8, 2, 4\}$	$\{0.7, 24, 0.1\}$	$\{19, 0, 2\}$
8	$\{10.2, 20, 0.1\}$	$\{9, 2, 4\}$	$\{0.7, 24, 0.1\}$	$\{22, 0, 2\}$
9	$\{11.3, 20, 0.1\}$	$\{19, 2, 4\}$	$\{4.4, 24, 0.1\}$	$\{37, 0, 2\}$
10	$\{10.2, 20, 0.1\}$	-	$\{0.7, 24, 0.1\}$	-

solver (see numbers in bold). Results for the platform configurations Φ_{CS}, and Φ_{DC} are depicted in Table 4 for the static segment. Table 5 shows results for Φ_{DC} for the dynamic segment.

4.3 Implementation and verification

We verified our design using a Hardware-in-the-Loop (HiL) setup which is common practice in the automotive industrial design process. A HiL can be considered as a form of real-time simulation where in addition to simulations, real hardware components are used. In our setting, the plants are simulated but real ECUs and a real FlexRay bus have been used. The purpose of a HiL system is to provide the necessary electrical stimuli, e.g., sensor values, which are required to realize the functionality of the application that is executed on real hardware components. Hence, our experiments consider all implementation platform details such as processors, driver stacks, buffers, and FlexRay configurations, that are required to study the impact of implementation effects on the control applications under simulation. Instead of implementing the full setup as depicted in Fig. 7 our equivalent HiL-setup essentially boils down to the consolidated architecture in Fig. 10 which consists of four ECUs, each with a 400MHz 32-Bit Embedded PowerPC and E-Ray FlexRay controllers, a bus monitoring unit to record the control outputs, and a FlexRay bus as the communication medium. We implement a FlexRay stack according to the FlexRay specification V2.1 [2] using commercial industrial design tools from SIMTOOLS [17] and prototyping FlexRay hardware from Elektrobit [18]. As depicted in Fig. 10, the tasks T_{CS}, and T_{CD} which are executed on ECU_{CS}, and ECU_{DC} simulate the plant equations for $\mathcal{C}_{CS}$ and $\mathcal{C}_{DC}$, i.e., T_{CS} simulates the sensor tasks T_1, T_2, T_3, T_4, and the actuator task T_6 of $\mathcal{C}_{CS}$, and T_{DC} simulates the sensor tasks T_7, T_8, and the actuator task T_{10} of $\mathcal{C}_{DC}$. Hence, T_{CS} computes the states x_1, x_2, x_3, x_4, which are sent via four messages according to $\Theta_1, \Theta_2, \Theta_3, \Theta_4$ to ECU6 where the control law is computed by T_5 and sent back to ECU_{CS} using Θ_5 and simulating

Table 5: Φ_{DC} for dynamic segment.

	Configuration I		Configuration II	
i	T_i of $\mathcal{C}_{DC}$	Θ_i of $\mathcal{C}_{DC}$	T_i of $\mathcal{C}_{DC}$	Θ_i of $\mathcal{C}_{DC}$
7	$\{7.2, 20, 0.1\}$	$\{46, 1, 4\}$	$\{5.8, 24, 0.1\}$	$\{65, 0, 2\}$
8	$\{7.2, 20, 0.1\}$	$\{47, 1, 4\}$	$\{5.8, 24, 0.1\}$	$\{95, 0, 2\}$
9	$\{12.3, 20, 0.1\}$	$\{47, 2, 4\}$	$\{7.7, 24, 0.1\}$	$\{233, 0, 2\}$
10	$\{7.2, 20, 0.1\}$	-	$\{5.8, 24, 0.1\}$	-

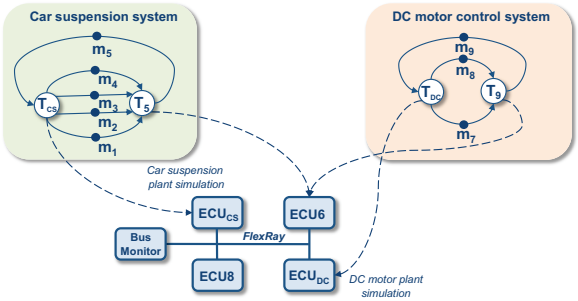

Figure 10: HiL setup for $\mathcal{C}_{DC}$ and $\mathcal{C}_{CS}$.

the actuation. Similarly, T_{DC} computes the states of the DC motor application that are sent over the bus to ECU6 where the control law is computed by T_9 and the control input is sent to T_{DC} for actuation. The task parameters $T_i \in \{o_i, p_i, e_i\}$, and message schedules $\Theta_i \in \{S_i, B_i, R_i\}$ have been selected according to the synthesized platform configurations depicted in Table 4 for both configurations and applications. For the experiments, we periodically introduced a step disturbance and observed the output y_1 and x_1, respectively, of $\mathcal{C}_{CS}$ and $\mathcal{C}_{DC}$ using the bus monitoring unit. Due to space constraints, we only show the plots for the static segment configurations according to the platform configurations in Table 4. The outputs for the dynamic segment can easily be carried out in a straightforward manner. Note that the outputs for the dynamic segment match the results of the static segment as the imposed freshness constraint $\tau = h$ remains the same. However, the platform parameters that realize the freshness constraint are different. As an example, the platform configurations Φ_{DC} for the dynamic segment are depicted in Table 5. It is interesting to observe that in the dynamic segment multiple messages sent by different ECUs may share the same slot, e.g., $\Theta_8 \in \{47, 1, 4\}$ and $\Theta_9 \in \{47, 2, 4\}$ in *configuration I*.

Output car suspension. The output y_1 of the car suspension system is depicted in Fig. 11 for both configurations. The reference value was set to $r = 0$, i.e., $r = 0$ indicates the nominal positions of the suspension system and the car body in absence of road disturbances. It can be seen from the figure that both applications are asymptotically stable as with $t \to \infty$, $y_1 \to 0$. Note that due to the fast sampling period $h_{CS} = 5ms$ the settling time is shorter in *configuration I* compared to *configuration II* where $h_{CS} = 12ms$. However, the *peak overshoot* for y_1 is much higher in the case of *configuration I*.

Output DC motor application. The output y_1 of the DC motor application is depicted in Fig. 12. In both configurations the system reaches the reference speed of $r = 50$ units in approximately same time and with equal peak values. This is because the sampling frequencies are quite close to each other in both configurations, i.e., $h_{DC} = 20ms$ in

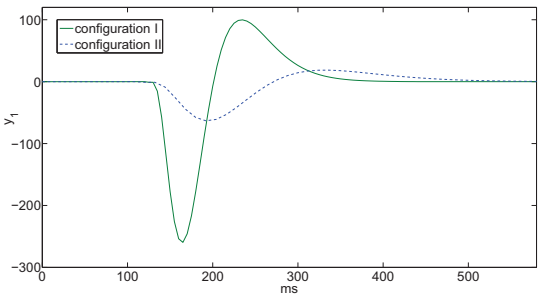

Figure 11: Output y_1 of $\mathcal{C}_{CS}$.

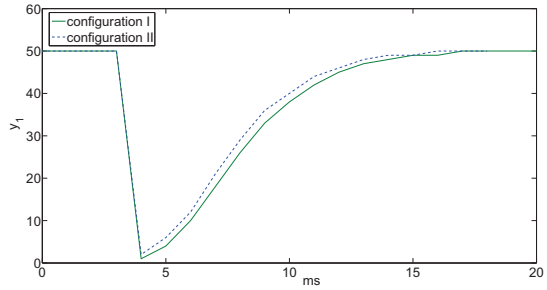

Figure 12: Output y_1 of $\mathcal{C}_{DC}$.

configuration I and $h_{DC} = 24ms$ in *configuration II*.

Constraint violation. Fig. 13 shows an example where a platform configuration Φ_{CS} violates the platform constraints in *configuration I*. In particular, the correlation constraint has been violated and the schedules for the two sensor messages m_3 and m_4 have been selected as $\Theta_3 = \{21, 0, 1\}$, and $\Theta_4 = \{22, 0, 1\}$ instead of the feasible values $\Theta_3 = \{13, 0, 1\}$, and $\Theta_4 = \{14, 0, 1\}$ as depicted in Table 4. As the controller task T_5 is triggered at $o_5 = 1.8ms$ (see Table 4), m_3 and m_4 are received by ECU6 after T_5 has been triggered. Hence, the control law is computed based on the actual values of x_1 and x_2 and the delayed values of x_3 and x_4, resulting in an unstable output of the plant.

5. CONCLUDING REMARKS

In this work, we proposed a joint design approach for automatically synthesizing controller gains along with their task and message schedules on distributed FlexRay platforms. Towards this, we first translated the timing constraints (or freshness constraints) imposed by the high level control goals into platform constraints. Next, the platform parameters such as task and message schedules were designed based on the derived platform constraints. Thus, the high level control-related timing constraints are respected at the implementation level. The constraints are solved by formulating a CSP and the formulation considers both the FlexRay static and dynamic segments. Our approach automates the design of platform parameters from given application level specifications. The proposed design method can be integrated with existing COTS design tools and significantly reduces the design effort required for complex distributed platforms. For future work, we plan to pursue the presented method for heterogeneous platforms and incorporate additional optimization objectives for the synthesis of optimal platform configurations.

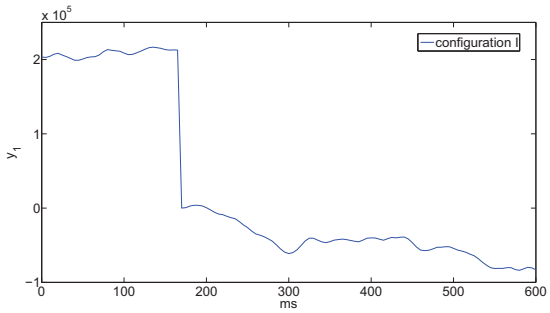

Figure 13: Unstable output y_1 of $\mathcal{C}_{CS}$.

6. REFERENCES

[1] W. Wolf. Cyber-physical Systems. *IEEE Computer*, 42(3):88 – 89, 2009.

[2] The FlexRay Communications System Specifications, Ver. 2.1. `www.flexray.com`.

[3] A. Schedl. Goals and Architecture of FlexRay at BMW . In *Slides presented at the Vector FlexRay Symposium*, 2007.

[4] J. Broy and K. D. Müller-Glaser. The Impact of Time-triggered Communication in Automotive Embedded Systems. In *Proc. of SIES'07*, pages 353–356, 2007.

[5] M. Grenier, L. Havet, and N. Navet. Configuring the Communication on FlexRay: The Case of the Static Segment. In *Proc. ERTS'08*, 2008.

[6] S. Reichelt, O. Scheickl, and G. Tabanoglu. The Influence of Real-time Constraints on the Design of FlexRay-based Systems. In *Proc. of DATE*, pages 858–863, 2009.

[7] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems*, 39:205–235, 2008.

[8] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. FlexRay Schedule Optimization of the Static Segment. In *Proc. of CODES+ISSS'09*, pages 363–372, 2009.

[9] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment. *IEEE Transactions on Industrial Informatics*, 7(99):1–1, 2011.

[10] T. Pop, P. Pop, P. Eles, and Z. Peng. Bus Access Optimisation for FlexRay-based Distributed Embedded Systems. In *Proc. of DATE'07*, pages 51–56, 2007.

[11] K. Schmidt and E.G. Schmidt. Schedulability Analysis and Message Schedule Computation for the Dynamic Segment of FlexRay. In *Proc. of VTC'10*, pages 1–5, 2010.

[12] H. Voit, R. Schneider, D. Goswami, A. Annaswamy, and S. Chakraborty. Optimizing Hierarchical Schedules for Improved Control Performance. In *Proc. of SIES'07*, pages 9 – 17, 2010.

[13] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems. In *Proc. of DATE'09*, pages 57–62, 2009.

[14] D. Deto, J.P. Lehoczky, L. Sha, and K.G. Shin. On Task Schedulability in Real-time Control Systems. In *Proc. of RTSS'96*, 1996.

[15] S. Samii, P. Eles, and Z. Peng. Design Optimization and Synthesis of FlexRay Parameters for Embedded Control Applications. In *Proc. of DELTA'11*, pages 66–71, 2011.

[16] D. Goswami, R. Schneider, and S. Chakraborty. Co-design of Cyber-Physical Systems via Controllers with Flexible Delay Constraints. In *Proc. of ASP-DAC*, pages 225–230, 2011.

[17] SIMTOOLS GmbH. `www.simtools.at`.

[18] Elektrobit. `www.elektrobit.com`.

[19] K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice Hall Upper Saddle River, NJ, 1996.

[20] Control Tutorials for Matlab. `http://www.engin.umich.edu/group/ctm`.

[21] Modeling a Car Suspension in States Space / State Space Systems with Matlab. `http://www.swarthmore.edu/NatSci/echeeve1/Class/e12/Lectures/SS/html/MatlabForSS.html`.

[22] Python-Constraint. `http://labix.org/python-constraint`.