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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2011 ACM 978-1-4503-0715-4/11/10 ...$10.00.

Keywords

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 schedulability 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.](image)

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

\[
\dot{x}(t) = A_c x(t) + B_c u(t),
\]

\[
y(t) = C_c x(t),
\]

(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
\[ x[k+1] = Ax[k] + Bu[k], \]
\[ y[k] = Cx[k], \] (2)
where
\[ A = e^{A \cdot h}, B = \int_0^h \left( e^{A \cdot t} \right) \cdot B_c, C = C_c. \] (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
\[ u[k] = Kx[k], \] (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 scheduler. Let a dispatch event for a task \( T_i \) (see Fig. 4) be defined by the tuple \( T_i = (o_i, p_i, c_i) \):

- 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.

**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-
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 pre-defined bus schedules. We refer to a bus schedule as the tuple \( \Theta \) defined bus schedules. We refer to a bus schedule as the tuple \( \Theta = \{ S_i, B_i, R_i \} \).

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 \( \Delta \) 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 \( \delta \ll \Delta \). 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 \( m_3 \) 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 is determined by the parameter \( p_{\text{LatestTx}} \).

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 \( T_i \), \( i \in \{\ldots, n\} \) and schedules for \( m \) messages triggered \( T_i \) is denoted by \( \Theta_i \). Thus, a control application \( C_i \) has a \((n + m)\)-tuple of platform parameters \( \Phi_i = \{T_i, \Theta_i\} \). We also call \( \Phi_i \) a valid platform configuration for \( C_i \). 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 \( C_i \) that is partitioned into \( n = 5 \) tasks that trigger \( m = 3 \) messages, as depicted in Fig. 4. Then the correspond-

![Figure 4: Scheduling of tasks and messages.](image-url)
of the feedback signals, i.e., realizing a constant delay of $\tau$, by choosing an appropriate platform configuration $\Phi_i$. As a result, we achieve a deterministic behavior of the controller $C_i$ 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

$$u[k] = Kx[k - 1].$$

(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 fresheness 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 $h = R T_{bus}$ 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

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

(9)

where $\lambda$ is a weight, $u(t)$ is the control input and $e(t) = \left[ y(t) - r \right]$ 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 $R_i$ for all control-related messages that realize

$$J_1^* = \min_{R_i} \left( \sum_{k=0}^{N} \int_{kh}^{(k+1)h} \left[ \lambda u(t)^2 + (1 - \lambda) e(t)^T e(t) \right] dt \right).$$

(10)

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

$$J_2^* = \min_{R_i} \frac{\xi}{R_i}.$$

(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 period, 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 $R_i$ 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 $\tau = h$.

#### Synchronicity constraint

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

$$\tilde{t}_i^k + \epsilon < t(\Theta_i), \quad \forall k \in N_0$$

(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 $\Theta_i = \{S_i, R_i, R_i\}$ as

$$t(\Theta_i) = B_i T_{bus} + k R T_{bus} + (S_i - 1) \Delta, \quad \forall k \in N_0.$$  

(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 $k R T_{bus}$ 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 $\epsilon$ 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

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

(14)

Note that (14) only depends on the bus schedule parameters $S_i, B_i \in \Theta$, 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

$$t(\Theta_i) = B_i T_{bus} + k R T_{bus} + N \Delta + (S_i - 1 - N) \delta, \quad \forall k \in N_0$$  

(15)

### Algorithm 1 Controller design under platform constraints.

Require: $T_{bus}$, $\lambda, A_i, B_i, C_i, e(0)$  
1: contSystem($A_i, B_i, C_i$)  
2: for all $i \in \{0, ..., 6\}$ do
3:   $R_i = 2^i$  
4:   $h = T_{bus} / R_i$  
5:   $c2d(A_i, B_i, C_i, h)$  
6:   $K = \text{computeGains()}$  
7:   $u = \text{computeControlLaw()}$  
8:   while simulate($x[0], N$) do
9:     $\text{getResponse()}$  
10:     for $N$ samples using (2) and (8)  
11:     $J_i = \text{getPerformance()}$  
12:   end while
13:   $J^*_i = \min_i J_i$.

Figure 5: Synchronicity and schedulability considering communication stack overhead. 


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 $m_1$. Using (6), (7), and (15) in (12) yields

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

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 $C_1$ 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 $T_{i,1}$, 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 $\tau$ 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 overwriten 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 $C_1$ are triggered at the same time instances $t_i^{k+1} = t_i^{k+2}$ (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 $T_{4,a} = \tau = h$ (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 $t_{i,a}^{k+1} - t_{i,a}^{k+2} = h$ which yields

$$o_i + (k + 1)h - (o_i + s + kh) = h, \forall k \in N_0, \quad (17)$$

and finally

$$o_i = o_i + s, \quad (18)$$

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 constraints are fulfilled for $T_i$ and $\Theta_i$, 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

$$o_i + c_i > \max (B_i T_{bus} + S_i, \Delta) + \epsilon. \quad (19)$$

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 $\Theta_i = \{S_{i,c}, R\}$

$$o_i + h > B_i T_{bus} + S_i, \Delta + \epsilon. \quad (20)$$

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

$$o_i + c_i > \max (B_i T_{bus} + (n_{i,s} + c_i)\delta) + N\Delta + \epsilon \quad (21)$$

and

$$o_i + h > B_i T_{bus} + (n_{i,c} + c_i)\delta + N\Delta + \epsilon \quad (22)$$

where the worst-case minislot consumption is bounded by

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

In (23), $c_j$ denotes the message size in terms of minislots. The first term $\sum_{j \in h_p} (c_j - 1)$ 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 $\Phi$ needs to satisfy the schedulability constraints (i) of all tasks $T_i \in \Phi_i$ that are executed on the ECUs, and (ii) all bus schedules $\Theta_i \in \Phi_i$ of the control-related messages. Let $T_1 \in T_1$ be the set of existing tasks that is mapped on ECU $N$. Further, let all periods be harmonic and multiples of the
to synthesize a feasible platform configuration $\Phi$ used to formulate a Constraint Satisfaction Problem (CSP) on the FlexRay bus. Then, the same slot is prohibited. Further, in the static segment, i.e., any intersection between bus cycles of schedules sharing dyanmic segment, different ECUs may share a slot in different tasks according to

$$T \left( i, j, k \right) \in \{ 0, \ldots, l \}.$$ (25)

and

$$H \in \{ 0, \ldots, m \}.$$ (26)

For any application tasks triggering or receiving messages, we consider a sufficiently large time window $\varepsilon$ 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 $i^k_i + \varepsilon$ 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 $i^k_i - \varepsilon$ according to (26).

Next, we apply the set of constraints derived in stage 2 which consists of

$$\begin{bmatrix} -\kappa_2 & -\kappa_2 & -\kappa_4 & -\kappa_4 \\ \kappa_2 & \kappa_2 & \kappa_4 & \kappa_4 \\ -\kappa_2 & \kappa_2 & -\kappa_4 & \kappa_4 \\ \kappa_4 & -\kappa_4 & \kappa_2 & -\kappa_2 \end{bmatrix}.$$ (31)

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

Further, the car suspension system model $C_{CS}$ is adapted from [21]

$$A_{DC} = \begin{bmatrix} 0 & 1 \\ \sigma_1 & \sigma_4 \end{bmatrix}, B_{DC} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C_{DC} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$ (32)

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.

### 4. EXPERIMENTAL RESULTS

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

$$A_{DC} = \begin{bmatrix} -\kappa_2 & -\kappa_2 & -\kappa_4 & -\kappa_4 \\ \kappa_2 & \kappa_2 & \kappa_4 & \kappa_4 \\ -\kappa_2 & \kappa_2 & -\kappa_4 & \kappa_4 \\ \kappa_4 & -\kappa_4 & \kappa_2 & -\kappa_2 \end{bmatrix}.$$ (33)

where $\sigma_1 = 100 \text{ kg}$, $\sigma_2 = 400 \text{ Ns/m}$, $\sigma_3 = 800 \text{ N/m}$, $\sigma_4 = 10 \text{ kg}$, and $\sigma_5 = 200 \text{ Ns/m}$, and $\sigma_6 = 800 \text{ N/m}$. 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, $C_{CS}$ consists of $n = 6$ tasks where each of the four sensor tasks $T_1, T_2, T_3$, and $T_4$ triggers a message $m$ on the FlexRay bus that needs to be scheduled according to $\Theta_1, \Theta_2, \Theta_3$, and $\Theta_4$. The control law is computed by the controller task $T_5$ which sends the control input to the actuator task $T_6$ via message $m_7$.

Further, $C_{DC}$ consists of $n = 4$ tasks where the two sensor tasks $T_7, T_8$ trigger $m_1$, $m_2$ according to $\Theta_5, \Theta_6$, respectively. The controller task $T_9$ computes the control input which is transmitted via $m_3$ corresponding to $\Theta_7$ to the actuator task $T_{10}$. Note that the controller tasks of both appli-
can be seen in Fig. 8 that the minimum cost for both configurations. It is interesting to observe that in the design of the controller gains different bus configurations as depicted in Table 1. Towards both objectives we evaluate transient performance of control input to avoid damage of the motor. To account for the suspension system is a fast response time to road disturbances and vibrations we are mainly interested in the transient performance of $C_{CS}$, and hence we choose (11) as the cost function to minimize the settling time $\xi$. The results for controller gains $K_{CS}$, sampling period $h_{CS}$, and repetition rate $R_{CS}$ are depicted in Table 2 for both configurations. It can be seen in Fig. 8 that the minimum cost $J_1^*$, i.e., the smallest settling time $\xi$, is achieved at $R = 1$, 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 $C_{DC}$ 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 $R$ for both configurations. It is interesting to observe that in

### Table 1: FlexRay bus configuration parameters.

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

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

<table>
<thead>
<tr>
<th>Design parameters</th>
<th>Configuration I</th>
<th>Configuration II</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_{CS}[ms]$</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>$R_{CS}$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$K_{CS}$</td>
<td>$[7712\ 2175\ -12141\ -61]$</td>
<td>$[594.9\ 189.6\ -1001.5\ -4.3]$</td>
</tr>
<tr>
<td>$h_{DC}[ms]$</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>$R_{DC}$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$K_{DC}$</td>
<td>$[-4.7197\ -0.5444\ -3.2687\ -0.4218]$</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8: Transient performance of $C_{CS}$ for configuration I and II.**

**Figure 9: Steady-state performance of $C_{DC}$ for configuration I and II.**

configuration I the available repetition rate at the minimum cost $J_1^*$ is $R_{DC} = 4$ and $h_{DC} = 20ms$, whereas in configuration II, $J_1$ is minimum at $R_{DC} = 2$, $h_{DC} = 24ms$ 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 $R$ 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 $R_{DC}$. Results for $K_{DC}, h_{DC}$, and $R_{DC}$ 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 $\pi$ 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 $\epsilon$ has been determined empirically, and hence might be conservative. Similarly, the worst-case execution times have been determined experimentally and considered with $\alpha_i = 0.1ms$ for any control-related task of $C_{CS}$ and $C_{DC}$. The finite domains for the offsets $\alpha_i$ have been discretized in 0.1ms step size, i.e., $\alpha_i \in \{0, 0.1, 0.2, \ldots\}$. We implemented the CSP for $C_{CS}$ and $C_{DC}$ 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
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 \( T_6 \) and the control input is sent to \( T_{DC} \) for actuation. The task parameters \( T_i \in \{a_i, p, e_i\} \) 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 \( C_{CS} \) and \( C_{DC} \) 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 \( \Phi_{DC} \) 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

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

<table>
<thead>
<tr>
<th>( C_{CS} )</th>
<th>Backtracking</th>
<th>Rec. Backtracking</th>
<th>Min. Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>320 sec</td>
<td>0.2 sec</td>
<td>1.5 sec</td>
</tr>
<tr>
<td>II</td>
<td>5584 sec</td>
<td>1.3 sec</td>
<td>81.4 sec</td>
</tr>
</tbody>
</table>

### Table 4: \( \Phi_{CS} \) and \( \Phi_{DC} \) for static segment.

<table>
<thead>
<tr>
<th>( i )</th>
<th>Configuration I</th>
<th>Configuration II</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( T_i ) of ( C_{CS} )</td>
<td>( \Theta_i ) of ( C_{CS} )</td>
</tr>
<tr>
<td>1</td>
<td>{0.5, 5.0, 1}</td>
<td>{11.0, 1}</td>
</tr>
<tr>
<td>2</td>
<td>{0.5, 5.0, 1}</td>
<td>{12.0, 1}</td>
</tr>
<tr>
<td>3</td>
<td>{0.5, 5.0, 1}</td>
<td>{13.0, 1}</td>
</tr>
<tr>
<td>4</td>
<td>{0.5, 5.0, 1}</td>
<td>{14.0, 1}</td>
</tr>
<tr>
<td>5</td>
<td>{1.8, 5.0, 1}</td>
<td>{24.0, 1}</td>
</tr>
<tr>
<td>6</td>
<td>{0.5, 5.0, 1}</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 5: \( \Phi_{DC} \) for dynamic segment.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( T_1 ) of ( C_{DC} )</th>
<th>( \Theta_1 ) of ( C_{DC} )</th>
<th>( T_0 ) of ( C_{DC} )</th>
<th>( \Theta_0 ) of ( C_{DC} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>{7.2, 20, 0.1}</td>
<td>{46, 1, 4}</td>
<td>{5.8, 24, 0.1}</td>
<td>{65, 0.2}</td>
</tr>
<tr>
<td>8</td>
<td>{7.2, 20, 0.1}</td>
<td>{47, 1, 4}</td>
<td>{5.8, 24, 0.1}</td>
<td>{95, 0.2}</td>
</tr>
<tr>
<td>9</td>
<td>{12.3, 20, 0.1}</td>
<td>{47, 2, 4}</td>
<td>{7.7, 24, 0.1}</td>
<td>{233, 0.2}</td>
</tr>
<tr>
<td>10</td>
<td>{7.2, 20, 0.1}</td>
<td>-</td>
<td>{5.8, 24, 0.1}</td>
<td>-</td>
</tr>
</tbody>
</table>

### Figure 10: HI  setup for \( C_{DC} \) and \( C_{CS} \).
Constraint violation. Fig. 13 shows an example where a platforms configuration $\Phi_{CS}$ 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 $o_5 = 1.8 \text{ms}$ (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.

6. REFERENCES


