# Optimizing Hierarchical Schedules for Improved Control Performance

Harald Voit*, Reinhard Schneider*, Dip Goswami*, Anuradha Annaswamy†, Samarjit Chakraborty*
*TU Munich, Germany, †Massachusetts Institute of Technology, USA

***Abstract*—Embedded control systems typically consist of several control loops, with different parts of each control application being mapped onto different processors that communicate over one or more communication buses. In such setups, the system architecture and scheduling policies have a significant impact on control performance. In this paper we show how to optimally choose the parameters of hierarchical schedules on the communication bus in order to improve multiple control performance metrics.**

## I. Introduction

As embedded systems become more complex and distributed – consisting of multiple processing units and communication buses – the gap between high-level control models and their actual implementations seem to widen. Control engineers are typically concerned with analyzing and simulating controllers based on well-defined semantics of both the plant and the controller being designed. Once a design is complete, has been analyzed and simulated, it is the task of the embedded systems engineer to come up with software implementations (e.g., in C) of the different control blocks (e.g., from MATLAB specifications) and implement the software on a hardware architecture or platform. By *platform* we mean the hardware on which the controller is implemented along with the operating system layer between the hardware and the software.

Often, the platform architecture consists of multiple processors connected by one or more communication buses. Further, multiple control applications share such a platform and need to be scheduled appropriately. A natural question that arises in such a setting is: *How does one quantify or account for the semantic gap between the control models and their implementations?* This is important because many of the assumptions on periodicity or zero-delay/jitter, that are common for control models, no longer hold in real-life implementations.

In this paper we consider multiple feedback controllers being implemented on a platform consisting of multiple processing units (PUs) that communicate over a shared bus. Each controller is split into multiple tasks that are then mapped onto different PUs. Hence, each PU consists of tasks from different control applications, and tasks from different PUs communicate over a single shared bus. Our goal is to develop a systematic method for scheduling the different message streams on the bus, in order to jointly maximize multiple control performance metrics (e.g., those related to stability cost, and transient and steady-state performance). Towards this, we focus on *hierarchical schedules* where each control application is allocated a Time Division Multiple Access (TDMA) slot (in order to provide temporal isolation between them), and message streams from the same controller are scheduled using fixed priority within the allocated TDMA slot. The problem is that of computing the values of the different parameters of this hierarchical scheduler in order to maximize control performance. Such parameters include the TDMA cycle length and the different slot sizes (with the priorities assumed to be constant).

It may be noted that our setup consists of *multiple controllers* and *multiple performance metrics*. Hence, certain choices of parameter values that improve the performance of one controller or one metric might reduce that of another, which leads to the *design space* being non-trivial. Further, although control performance improves monotonically with decreasing the delay experienced by the different message streams (at least for the controllers we study here), the *rate* of improvement is not constant. Hence, identifying the sweet spot that leads to optimal control performance is in general a non-trivial optimization problem (whose complexity would depend on both the controllers, the performance metrics and the underlying architecture).

For the platform architecture and scheduling policies studied here, we derive a closed-form formulation of message delay, as a function of the various scheduler parameters. This is used in conjunction with three control performance metrics in order to identify optimal scheduling parameters. Further, we consider a parameterized joint performance metric, which is a linear combination of the three different metrics. Our simulation results show that the optimal choice of scheduler parameters heavily depend on the parameters of the joint performance metric. This illustrates the *control-scheduler co-design* issue that is growing in importance as embedded systems architectures become more complex and tunable.

The problem of bridging the gap between the semantics of control models and that of the implementation platform has of late attracted a lot of attention, for example in [1]-[9]. Ngheim *et al.* in [1] and Yazarel *et al.* in [2] have quantified this exact gap/error for linear controllers implemented on time-triggered platforms. In [3], [4], the authors address the problem of co-designing controllers and the underlying schedulers as in this paper. In contrast to our work, where we derive closed-form expressions of message delay from the scheduler parameters and use these delay values to estimate control performance, the authors of [3], [4] resort to simulation to estimate the distribution of message delay values and use this distribution

978-1-4244-5841-7/10/$26.00 ©2010 IEEE

as an input to the Jitterbug toolbox [5] to compute control performance. Also, in [3] only one of the three performance metrics that we study here is addressed. Related papers such as [6]-[8] address the estimation of scheduler periods or priorities with the aim of improving control performance. In cases, such as in [7], analytical solutions have also been found. A delay impulsive model is proposed in [9] to represent a distributed control implementation, and is realized using a FlexRay architecture. However, the realistic effects of a FlexRay based implementation such as jitter and sporadic messages are not addressed in [9]. Our contribution, compared to [1]-[9] is the use of multiple control performance metrics and the use of suitable approximations to account for the effects of message delays. Further, we use a general model of the implementation platform that enables us to compute the delays encountered in heterogeneous architectures in a form that can be plugged into our control performance model. Finally, while previous studies were only concerned with relatively simple schedules, our work shows how complex bus arbitration schemes – like hierarchical schedules – may be optimized or improved control performance.

The rest of the paper is organized as follows. We next give a short overview of the feedback control systems studied here, along with a description of our delay model (Section II). This is followed by our analysis of communication architectures, where we derive a closed-form expression for message delay as a function of scheduler parameters and the timing properties of incoming message streams (Section III). Next, we propose our method for control-scheduling co-design, i.e., how to determine optimal scheduler parameters for specific parameter valuations of the joint control performance metric (Section IV). We then present our simulation results to illustrate our hypothesis – the need for controller- and performance metric-specific architectures/schedulers (Section V) followed by some directions for future work (Section VI).

## II. Feedback Control Systems

Several engineering systems employ the use of automatic control to improve their performance in terms of speed, accuracy, cost, and efficiency. Typically, this requires the use of sensors to monitor the system performance, actuators to introduce corrections to the system behavior, a reference command that indicates the desired behavior, and controllers that compute the timing and magnitude of these corrections (see Fig. 1 for a schematic). The field of control theory and engineering is dedicated to the analysis and synthesis of various control methods that determine the control structure for a given system.

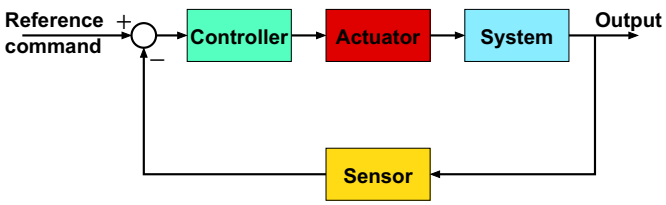


Fig. 1. Schematic Feedback Control System

A typical design procedure begins with the determination of the underlying model of the system to be controlled. Many physical systems can be described with ordinary differential equations (ODE) in continuous time. These ODEs may be based on conservation equations and constitutive relations. In some cases, difference equations are used to describe the system, and may be derived using a *System-Identification* based methodology that consists of evaluating input-output responses and determining a discrete-time system that best fits these responses. Similar to the system-model, an actuator model is determined as well. Once these models are derived, a controller is designed using state-space [10], frequency-domain [11], or geometric methods [12]. The success of the control performance is not only determined by the accuracy of the system models and the specific control method used, but also – as described in the previous section – on the accuracy of modeling the implementation architecture. In order to lead towards an implementation platform that successfully meets the requirements of a feedback control system, we also model the effects of control implementation. This is addressed in what follows.

As mentioned in the previous section, our platform architecture consists of several processing elements connected by a shared communication bus. Such architectures introduce implementation errors of different kinds [13]. For the purposes of our discussion here, we model the implementation errors in the form of a delay and a disturbance (see Fig. 2) that represent the effect of sampling, scheduling, discretization and quantization errors. We analyze the control performance of the feedback system in Fig. 2 in order to lead to a control-platform co-design.

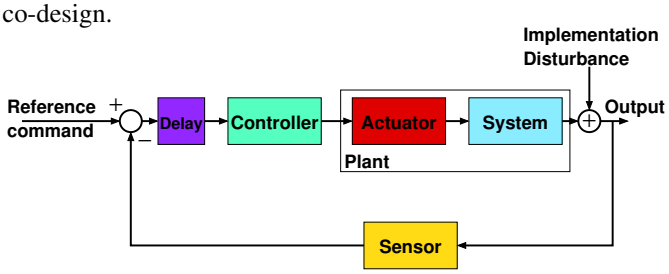


Fig. 2. Schematic Feedback Control System with Time Delay

The two main properties that need to be addressed while discussing Quality of Control are stability and performance. Stability is concerned with the well-behavedness of the system signals in the presence of feedback control. In particular, it is desired that the system signals do not exceed a certain bound. A stricter measure of stability is concerned with the regulation of the underlying error signals to zero.

Closed-loop control performance objectives typically include tracking of a command signal in addition to rejecting or minimizing the effects of disturbances, measurement noise, and modeling errors. Optimizing various performance objectives over the set of stabilizing controllers is the main thrust of optimal control methods, such as $L_1$, $H_2$, and $H_\infty$ control. Fundamental limits of closed-loop control performance for general dynamic systems, for command following as well as

disturbance rejection, can be computed using Bode Sensitivity Integrals [14], [15], [16]. For ease of exposition, we limit our discussion in this paper to performance metrics that pertain to stabilizing controllers that ensure satisfactory command following for lower order systems.

## III. Communication Analysis

We consider a platform architecture with multiple distributed control applications (see Fig. 3). The control applications are partitioned into a number of tasks that are mapped onto different PUs. The PUs communicate via a shared communication bus and run different tasks from one or more control applications. Scheduling on the processors and arbitration on the shared bus introduces delays in various control signals. These delays are dependent on the arbitration/scheduling policy on the buses and on the PUs.

### A. System Description

We now briefly describe the platform architecture, the mapping of various control tasks on it, and the hierarchical scheduling policy on the shared bus. For the ease of illustration, this setup will form the basis of the techniques we propose. However, they hold true for other setups as well.

**Hardware/Software architecture:** We consider three control applications: *controller 1, 2 and 3* shown in Fig. 4, 5 and 6. Each controller is connected to an actuator and a sensor. The aim of each control application is to compute the control input to the plant such that the closed-loop system meets desired performance. Towards this, the controllers read sensors values to determine the state of the closed-loop system and compute commands based on the deviation from the desired performance. Control applications are partitioned into a number of tasks, which are mapped onto different PUs connected via the communication bus. In our architecture (see Fig. 3) PU1 hosts the tasks responsible for reading the reference command from the user, PU2 hosts all tasks that compute control commands, PU3 hosts the tasks responsible for reading sensors, and the tasks responsible for providing plant commands are mapped onto PU4.

**Control applications:** *Control application 1* (Fig. 4) is partitioned into four tasks: $T_1$, $T_2$, $T_3$ and $T_4$. Task $T_1$ reads sensor $S_1$ and sends sensor signal $m_1$ via the communication bus to the task $T_3$. Similarly, task $T_2$ reads the reference command from the user and sends the reference command $m_2$ via the communication bus to task $T_3$. The control input is computed in $T_3$ using the messages $m_1$ and $m_2$. The processed output of task $T_3$ is sent via the communication bus to task $T_4$. The plant $P_1$ receives the control input from the task $T_4$. Fig. 4 (a) and (b) show the *task graph* and the *closed-loop block diagram* for *controller 1*.

*Control application 2* (Fig. 5) is partitioned into three tasks: $T_5$, $T_6$ and $T_7$. Task $T_5$ reads the sensor $S_2$ and sends the sensor signal $m_5$ via the communication bus to the task $T_6$. Controller task $T_6$ computes the control command based on the sensor feedback signal and the constant predefined reference command. The task $T_6$ sends the control command to task $T_7$ via the communication bus. The plant $P_2$ receives input signal from the task $T_7$. Fig. 5 (a) and (b) show the *task graph* and the *closed-loop block diagram* for *controller 2*.

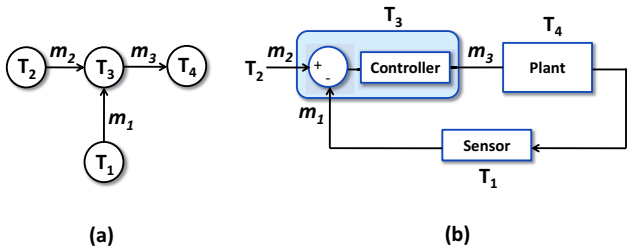


Fig. 4. *Controller 1*: (a) The Task Graph (b) Closed-Loop Block Diagram

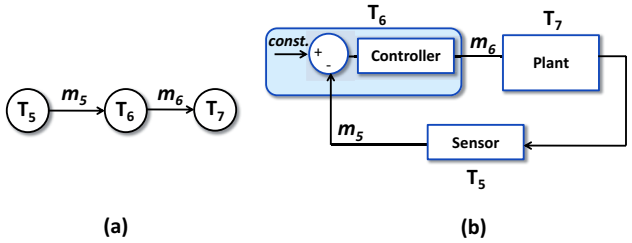


Fig. 5. *Controller 2*: (a) The Task Graph (b) Closed-Loop Block Diagram

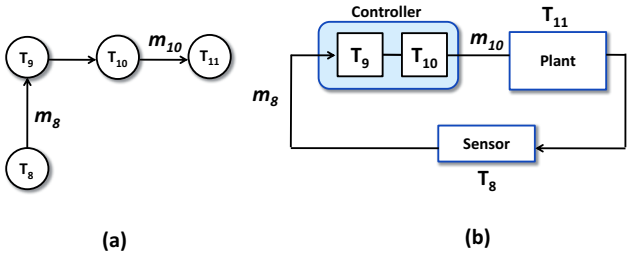


Fig. 6. *Controller 3*: (a) The Task Graph (b) Closed-Loop Block Diagram

*Control application 3* (Fig. 6) is partitioned into four tasks: $T_8$, $T_9$, $T_{10}$ and $T_{11}$. Task $T_8$ reads the sensor $S_3$ and sends the sensor signal $m_8$ via the communication bus to the task $T_9$. Task $T_9$ computes one part of the control signal and sends the resulting command to the task $T_{10}$. Task $T_{10}$ computes the control command $m_{10}$ and sends it via the communication bus to the task $T_{11}$ which sends the control command to the plant $P_3$. Fig. 6 (a) and (b) show the *task graph* and the *closed-loop block diagram* for *controller 3*.

**Communication scheduling:** The *shared communication bus* follows a hierarchical TDMA/FP scheduling policy (Fig. 7). At the top-level of the scheduler runs a TDMA scheduler, i.e., the communication bandwidth is divided into equal cycles of length $c$. The TMDA cycles are further divided into three slots, i.e., $s_1$, $s_2$ and $s_3$ which are assigned to the *control applications 1, 2* and *3*. Therefore, *control application i (i=1, 2, 3)* can only send input streams or messages in slot

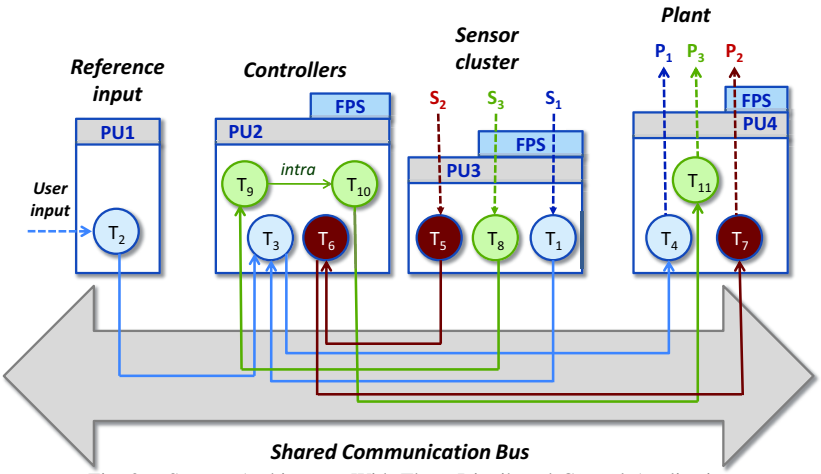


Fig. 3. System Architecture With Three Distributed Control Applications

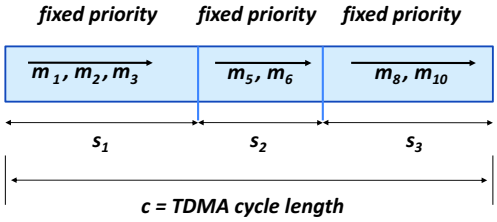


Fig. 7. The Communication Bus: Hierarchical TDMA/FP Scheduler

$s_i$. For example, *control application 1* can only access the communication bus in $s_1$. Further, all the streams or the messages coming into/from the *control application* which are transmitted via the communication bus follow a fixed priority scheduler (FPS). For example, *control application 1* exchanges three messages among its various tasks: $m_1$, $m_2$ and $m_3$. These messages can only be transmitted during the slot $s_1$. If all the three messages are ready to be transmitted via $s_1$ then first $m_1$ gets access to the bus. Subsequently, $m_2$ and $m_3$ are transmitted respectively.

### B. Compositional timing analysis

In this section we will give an overview of the *real-time calculus* (RTC) framework [17], [18], which is an analytical method for analyzing performance properties of distributed embedded real-time systems. Given several distributed applications that are mapped onto different processing and communication resources, e.g., in the architecture depicted in Fig. 3, we are now interested in computing the maximum end-to-end delay experienced by each of the *control applications*.

In the following, we refer to an abstract system representation (depicted in Fig. 8) to illustrate the applicability of the timing analysis framework. The system consists of two processing units PU1 and PU2 and a shared communication medium. The processing units execute the control tasks $T_1$, $T_2$ and $T_3$ and transmit the processed output over the shared bus. The input of a task can be represented by an arrival pattern $A(t)$ of a data stream, i.e., sensor readings. The cumulative function $A(t)$ denotes the total number of events that arrive

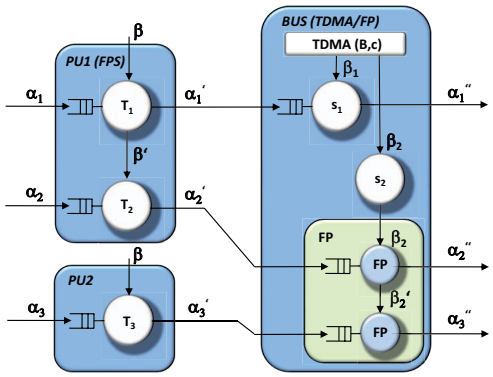


Fig. 8. System With Hierarchical Bus Scheduling

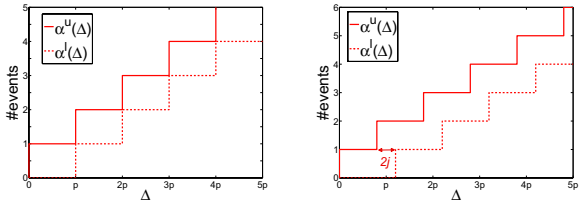


Fig. 9. a) Periodic Arrival Curves. b) Periodic With Jitter j.

during the time interval $(0, t]$. However, the number of events that may arrive in *any* time interval of length $\Delta$ that trigger a task execution or a message transmission can be upper- and lower-bounded by the pair of *arrival curves* $\alpha = (\alpha^u, \alpha^l)$. Formally, we obtain the following mathematical inequality:

$$\forall \Delta \geq 0,\ \forall t \geq 0 : \alpha^l(\Delta) \leq A(\Delta + t) - A(t) \leq \alpha^u(\Delta) \quad (1)$$

In other words, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ denote the maximum and minimum number of events that can arrive within *any* time interval of length $\Delta$. Thus, using this event model, we can represent the timing properties of standard event models - like *periodic, periodic with jitter* and *sporadic* - as well as arbitrary arrival patterns by an appropriate choice of $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$. Fig. 9 (a) illustrates an example for the upper and lower arrival curves representing a strictly periodic event stream with period $p$. Fig. 9 (b) depicts a pair of arrival curves with period $p$ and

jitter $j$, i.e., in a periodic sensor stream with jitter the sensor samples arrive at an average time interval of $p$ time units, but can deviate from the ideal periodic arrival.
Similarly, we can model the available resource capacities on a processor or a bus to process a task or transmit a message. The cumulative function $C(t)$ captures the number of events that can be processed in $(0, t]$. The service available to a task or message is upper- and lower-bounded by the pair of *service curves* $\beta = (\beta^u, \beta^l)$. Thus, the following mathematical inequality hold true:

$$\forall \Delta \geq 0,\ \forall t \geq 0 : \beta^l(\Delta) \leq C(\Delta + t) - C(t) \leq \beta^u(\Delta) \quad (2)$$

Hence, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ denote the maximum and minimum number of events that can be processed within *any* time interval of length $\Delta$. The service curves can also be expressed in terms of the maximum and minimum number of resource units (e.g., processor cycles, execution times) that are required to process an event within any $\Delta$. For this purpose, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ are scaled with the execution requirement demanded by a task or message due to each activation. When running several tasks on a shared resource, e.g., $T_1$ and $T_2$ on $PU1$, the bounds on the service available to the tasks using this resource depend on the scheduling policy being used. On $PU1$ the tasks are executed according to FPS. Hence, the full service $\beta$ is available to the highest priority task $T_1$ to process the input stream $\alpha_1$. Consequently, the remaining service $\beta'$ after processing $\alpha_1$ serves as an input for the execution of $T_2$ processing $\alpha_2$. Further, the processed data streams $\alpha' = (\alpha^{u'}, \alpha^{l'})$ and the remaining service $\beta' = (\beta^{u'}, \beta^{l'})$ can be computed by the following equations:

$$\alpha^{u'} = \min\big\{\big(\alpha^u \otimes \beta^u\big) \oslash \beta^l,\ \beta^u\big\} \quad (3)$$

$$\alpha^{l'} = \min\big\{\big(\alpha^l \oslash \beta^u\big) \otimes \beta^l,\ \beta^l\big\} \quad (4)$$

$$\beta^{u'} = \big(\beta^u - \alpha^l\big)\ \overline{\oslash}\ 0 \quad (5)$$

$$\beta^{l'} = \big(\beta^l - \alpha^u\big)\ \overline{\otimes}\ 0 \quad (6)$$

Let $\overline{\mathbb{R}} := \mathbb{R} \cup \{+\infty, -\infty\}$ and $\mathcal{F} = \{f : \mathbb{R}_+ \to \overline{\mathbb{R}} \mid \forall s < t,\ 0 \leq f(s) \leq f(t)\}$. The (min,+) convolution $\otimes$ and deconvolution $\oslash$ operators are defined by: $\forall f, g \in \mathcal{F}$ and $\forall t \in \mathbb{R}_+$,

$$\big(f \otimes g\big)(t) = \inf\big\{f(s) + g(t-s) \mid 0 \leq s \leq t\big\}$$
$$\big(f \oslash g\big)(t) = \sup\big\{f(t+u) - g(u) \mid u \geq 0\big\}$$

The (max,+) convolution $\overline{\otimes}$ and deconvolution $\overline{\oslash}$ operators are defined by: $\forall f, g \in \mathcal{F}$ and $\forall t \in \mathbb{R}_+$,

$$\big(f \overline{\otimes} g\big)(t) = \sup\big\{f(s) + g(t-s)\ \mid\ 0 \leq s \leq t\big\}$$
$$\big(f \overline{\oslash} g\big)(t) = \inf\big\{f(t+u) - g(u)\ \mid\ u \geq 0\big\}$$

Considering a set $S \subseteq \mathcal{F}$, the *supremum* (sup) is defined by the smallest $U \in \mathcal{F}$ such that $x \leq U$, $\forall x \in S$. Similarly, the *infimum* (inf) of $S$ is the largest $L \in \mathcal{F}$ such that $x \geq L$, $\forall x \in S$. Given $\alpha^u(\Delta)$ and $\beta^l(\Delta)$ the maximum backlog $b$ at the input buffer, and the maximum delay $d$ that is experienced by the event stream $\alpha$ can be computed as follows:

$$b = \sup\big\{\alpha^u(\Delta) - \beta^l(\Delta)\ \mid\ \Delta \geq 0\big\} \quad (7)$$

$$d = \sup\big\{\inf\big\{\tau \geq 0 \mid \alpha^u(s) \leq \beta^l(s+\tau)\big\} \mid s \geq 0\big\} \quad (8)$$

Note that (7) and (8) denote the maximum vertical and horizontal deviations between $\alpha^u(\Delta)$ and $\beta^l(\Delta)$.

The processed data streams ${\alpha_1}', {\alpha_2}'$ and ${\alpha_3}'$ trigger messages to be transmitted on the shared bus according to a *hierarchical* TDMA/FP scheduling policy. The top-level scheduler is modeled as a TDMA resource with a total bandwidth of $B$ and a cycle length of $c$. Within every TDMA cycle the time slot $s_i$ is assigned to one *control application*. In Fig. 8, slot $s_1$ is assigned to the data stream ${\alpha_1}'$ whereas slot $s_2$ is assigned to ${\alpha_2}'$ and ${\alpha_3}'$. The service bounds for a TDMA resource can be modeled as follows [19]:

$$\beta_i^l = B \max\left\{\left\lfloor \frac{\Delta}{c} \right\rfloor s_i, \Delta - \left\lceil \frac{\Delta}{c} \right\rceil (c - s_i)\right\} \quad (9)$$

$$\beta_i^u = B \min\left\{\left\lceil \frac{\Delta}{c} \right\rceil s_i, \Delta - \left\lfloor \frac{\Delta}{c} \right\rfloor (c - s_i)\right\} \quad (10)$$

During the time interval $\Delta = (c - s_i)$ no service is guaranteed to any data stream that is assigned to any slot $s_i$. According to the *hierarchical* scheduling policy the service available to the data streams assigned to any time slot $s_i$ is determined according to the FPS. The full service $\beta_1$ is available for transmitting the data stream ${\alpha_1}'$. The input streams ${\alpha_2}'$ and ${\alpha_3}'$ share slot $s_2$ providing the service $\beta_2$ where ${\alpha_2}'$ is assigned a higher priority than ${\alpha_3}'$. Thus, the full service $\beta_2$ is available to ${\alpha_2}'$ while the remaining service ${\beta_2}'$ is available for transmitting ${\alpha_3}'$ on the bus. Now, the output event streams ${\alpha_1}'', {\alpha_2}''$ and ${\alpha_3}''$ can serve again as an input to further processing units, e.g., executing actuator tasks.

Considering the system in Fig. 8, the end-to-end delay can be computed as the sum of the individual delays at the different processing components (due to task processing and bus communication) experienced by an input data stream, e.g., the data stream $\alpha_1$ experiences a delay $d_1$ due to task processing of $T_1$ and another delay $d_2$ due to the message transmission on the bus. Thus, using (8), the total delay experienced by $\alpha_1$ is computed as $\tau = d_1 + d_2$. Hence, using the compositional performance model presented in this section, we are able to compute the maximum end-to-end delay $\tau$ experienced by any message along the path from the sensor to the actuator, of any distributed *control application*. For example, in Fig. 3 the worst case end-to-end delay experienced by *control application 3* is computed by $\tau = d_{T_8} + d_{m_8} + d_{T_9} + d_{T_{10}} + d_{m_{10}} + d_{T_{11}}$.

## IV. Co-design

### A. Design space exploration

In the previous section, we discussed the analysis of timing properties in distributed embedded systems, e.g., the maximum end-to-end delay experienced by a data stream. Given an

architecture such as in Fig. 3, we are interested in exploring every possible system configuration within a specified range of interest. Towards this, we generate the system configurations while varying the TDMA cycle length $c$ and the slot size $s_i$ assigned to any control application $C_i$ according to Alg. 1 (lines 1 and 2). Towards exploring all the possible system configurations for a particular control application $C_i$, we vary the slot size $s_i$ that is assigned to this controller while keeping the slot sizes for the other controllers $C_j$ fixed (and equal to the maximum size of the messages being transmitted in those slots). For every such system configuration, we compute the maximum end-to-end delay $\tau$ encountered by every control applications (line 3) according to the timing analysis technique described in Section III. If the delays encountered by all three controllers are *finite*, we include the corresponding system configuration into the *set* of *feasible* system configurations $\Omega$ (lines 4 and 5). Therefore, at the end of the design space exploration for a particular controller $C_i$, we evaluate a set of *feasible* system configurations with corresponding TDMA bus parameters.

**Algorithm 1** Design space exploration of system configurations.

```
Require: C_i, C_j, c_min, c_max, s_min, s_max, step_c, step_s
 1: for c ∈ [c_min + n × step_c ≤ c_max], n ∈ [0, 1, 2...] do
 2:     for s_i ∈ [s_min + k × step_s ≤ s_max], k ∈ [0, 1, 2...] do
 3:         τ = computeDelay(), ∀C_i, C_j
 4:         if τ ∈ [0, ∞), ∀C_i, C_j then
 5:             add configuration to Ω //set of feasible configurations
 6:         else
 7:             discard configuration
 8:         end if
 9:     end for
10: end for
```

### B. Performance Metrics

Typical measures of stability of a feedback system are gain and delay margins. The latter is our focus here, using which we define a stability cost function $P_0$ as

$$\mathcal{P}_0 = \frac{1}{L_m} \tag{11}$$

where $L_m$ is the delay margin of the plant $P$ with a controller $C$. The delay margin $L_m$ is the amount of time delay which can be tolerated by the system before it gets unstable.

We select two different performance metrics, one that quantifies transient performance and another that quantifies steady-state performance, both in the context of following a reference command. A measure of transient performance is Peak Overshoot [11], and can be defined as follows: Defining $g(t)$ as the step response, and $t_0$ is the first time-instant when $g(t)$ achieves its steady-state value $g_{ss}$, the peak overshoot cost, denoted as $\mathcal{P}_1$, is defined as

$$\mathcal{P}_1(e) = \begin{cases} \max(g(t)) \text{ for } t > t_0 & \text{if } t_0 \text{ exists} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

A steady-state performance metric of command-tracking can be defined using integral cost function of the tracking error. Denoted as $\mathcal{P}_2$, this performance metric is defined as

$$\mathcal{P}_2 = \int_0^\infty e(t)^2 dt \tag{13}$$

where $e(t)$ is the tracking error, and is denoted as Cheap Control [20].

Overall, a desired controller can be defined as one that minimizes the cost function $J$ defined as

$$J = \sum_{i=0}^{2} \lambda_i \mathcal{P}_i \tag{14}$$

where $\lambda_i$s are suitably chosen weights. It is clear from (14) that equal values of $\lambda_i$ imply an equal weighting in stability and performance, while a larger value of $\lambda_2$ (and $\lambda_3$) relative to $\lambda_1$ implies a greater emphasis on transient (and steady-state) performance. It should be noted that the above choices for performance metrics are typical examples, and reflect measures of stability and performance costs associated with command tracking. Similar costs based on disturbance rejection, control-energy, and modeling errors can also be included in $J$. A typical cost function used in optimal control, for instance, not only includes tracking error $e$, but also a function $u^2$ to reflect the actuation energy introduced for control [3].

### C. Computation of Performance Metrics

We illustrate the properties of a feedback control system described above through a specific example. We assume that the plant in Fig. 1 can be described using a linear ordinary differential equation with a transfer function $P(s)$. The system transfer function $P(s)$ is given by

$$P(s) = \frac{K_P}{s^2 + s} \tag{15}$$

and a controller-model

$$C(s) = K(s + b)e^{-\tau s} \tag{16}$$

where $\tau > 0$ is the time-delay due to computation and communication. We consider a negative unit feedback structure. The closed-loop transfer function is then given by

$$G(s) = \frac{K_P K(s + b)e^{-\tau s}}{s^2 + s + K(s + b)e^{-\tau s}} \tag{17}$$

The delay margin $L_m$ is determined by the phase of the forward-loop system of $G(s)$ at the crossover frequency, $\omega_c$, where the forward-loop gain is unity. For the system given in (15) and (16), this can be computed analytically as

$$\begin{aligned} &\omega_c(K, b, K_P) = \\ &\Big(-\frac{1}{2} + \frac{K_P^2 K^2}{2} \\ &+ \frac{1}{2}\sqrt{1 - 2K_P^2K^2 + 4K_P^2b^2K^2 + K_P^4K^4}\Big)^{\frac{1}{2}} \end{aligned} \tag{18}$$

With $\omega_c$ we can compute the phase margin $\varphi_m$

$$\begin{aligned} &\varphi_m(K, b, K_P) = \\ &\arctan\Big(-\frac{1}{2}(b-1)(K_P^2K^2 - 1 + \xi, \\ &- \frac{1}{2\sqrt{2}}\sqrt{K_P^2K^2 - 1 + \xi}(K_P^2K^2 + 2b - 1 + \xi)\Big) \\ &+ \pi \end{aligned} \tag{19}$$

where

$$\xi = \sqrt{1 + 2K_P^2K^2(0.5 \cdot K_P^2K^2 + 2b^2 - 1)} \quad (20)$$

is used for better readability. The delay margin is then given by

$$L_m(K, b, K_P) = \frac{\varphi_m(K, b, K_P)}{\omega_c(K, b, K_P)} \quad (21)$$

and $\mathcal{P}_0$ as defined in (11) is consequently given by

$$\mathcal{P}_0(K, b) = \frac{\omega_c(K, b, K_P)}{\varphi_m(K, b, K_P)} \quad (22)$$

The determination of the peak overshoot $\mathcal{P}_1$, defined as in (12) is done numerically. In order to obtain analytical results for $\mathcal{P}_2$, the time delay in (16) is replaced by a 4th-order Padé approximation given by

$$\begin{aligned} e^{-\tau s} &\approx p_4(-\tau, s) \\ &= \frac{\tau^4 s^4 - 20\tau^3 s^3 + 180\tau^2 s^2 - 840\tau s + 1680}{\tau^4 s^4 + 20\tau^3 s^3 + 180\tau^2 s^2 + 840\tau s + 1680} \end{aligned} \quad (23)$$

Using (23), approximate closed-loop transfer function $\tilde{G}(s)$ can be derived as

$$\tilde{G}(s) = \frac{K(s+b)p_4(-\tau s)}{s^2 + s + K(s+b)p_4(-\tau s)} \quad (24)$$

Using (24), which is an approximation of the controller together with the system, the computation of $\mathcal{P}_2$ defined in (13) can be carried out in a straightforward way. $\mathcal{P}_2$ can be computed for general closed-loop systems using Bode Sensitivity Integrals [20].

### D. Optimal Co-design

Using the performance metrics computed above, we now outline an optimization procedure that results in a minimal $J$. This procedure will not only carry out an optimization over all control parameters in $C(s)$ but also an implementation optimization over all possible configurations.

We represent the cost $J$ in (14) using the notation $\mathcal{P}_{k,i}^j$ for the performance metrics, where index $k = 0, 1, 2$ corresponds to the three different performance metrics $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$, $i = 1, 2, 3, \ldots, |\Omega|$ represents the $|\Omega|$ feasible system configurations, and $j = 1, 2, 3, \ldots, m$ represents $m$ copies of the controller in the embedded system architecture. The goal is to find a feasible configuration and a vector of controller parameters, such that the overall cost $J$ is minimized and therefore the corresponding QoC is maximized. In order to find this optimal value we consider the total cost $\bar{\mathcal{P}}_i^j$ for a given controller $j$ and a configuration $i$, which is defined as

$$\bar{\mathcal{P}}_i^j(\theta_j) = \sum_{k=0}^{2} \lambda_k \mathcal{P}_{k,i}^j(\theta_j) \quad (25)$$

where $\theta_j$ represents the parameters of the controller $j$.

The total cost over all controllers, for a given configuration $i$, denoted as $\tilde{\mathcal{P}}_i(\theta_j)$ is given by

$$\tilde{\mathcal{P}}(\theta_j) = \sum_{j=1}^{m} \bar{\mathcal{P}}_i^j(\theta) \quad (26)$$

The overall optimal cost can now be defined as

$$\mathcal{P}^*(\theta^*, i^*) = \min_{\substack{i=1,\ldots,|\Omega| \\ \theta \in \Gamma}} (\tilde{\mathcal{P}}_i(\theta)) \quad (27)$$

where $\mathcal{P}^*$ is the minimum cost obtained for an optimal configuration $i^*$ and optimal control parameter $\theta^*$ and $\Gamma$ denotes the set of all feasible controller parameters.

In summary, for a plant given in (15) and controller in (16), performance metrics $\mathcal{P}_0$, $\mathcal{P}_1$ and $\mathcal{P}_2$ can be computed using (22), (12), and (13), respectively, for a given $\tau$ in an analytical manner. It should also be noted that for the class of plants as in (15), both performance metrics $\mathcal{P}_1$ and $\mathcal{P}_2$ deteriorate monotonically with $\tau$. $\mathcal{P}_0$ on the other hand is independent of $\tau$, and represents the delay margin. This implies that the overall optimal cost $\mathcal{P}^*$ represents the worst case performance and that any configuration that yields a smaller set of delays will only lead to an improved performance.

## V. Results

Given the system architecture shown in Fig. 3, we choose sensor signals to be *periodic* and the remaining sensors to be *periodic with jitter*. We assumed that $S_1$ sends the sensor reading periodically at an interval of 20 ms and that $S_2$ and $S_3$ send the sensor streams periodically with periods 30 and 40ms respectively with jitters 6 and 2 ms. Our architecture shown in Fig. 3 has 11 tasks mapped onto various PUs and are executed according to FP schedulers. The execution demand of the tasks are (in ms): $T_1 = 2$; $T_2 = 2$; $T_3 = 3$; $T_4 = 3$; $T_5 = 1$; $T_6 = 2$; $T_7 = 4$; $T_8 = 1$; $T_9 = 2$; $T_{10} = 3$ and $T_{11} = 6$. The 11 tasks generate 7 messages which are transmitted via the shared communication bus according to a *hierarchical* TDMA/FP scheduling policy. The transmission times of the messages are (in ms): $m_1 = 4$; $m_2 = 2$; $m_3 = 2$; $m_5 = 3$; $m_6 = 2$; $m_8 = 2$; $m_{10} = 2$.

For the design space exploration, we vary the TDMA cycle length $c$ from $c_{min} = 10$ms to $c_{max} = 40$ms in steps of $step_c = 5$ms according to Alg.1. While varying the TDMA cycle length, we also vary the slot size $s_1$ of controller $C_1$ we are interested to optimize, while keeping the other two slot sizes $s_2$ and $s_3$ fixed, i.e., we vary $s_1$ from $s_{min} = 1$ ms to $s_{max} = (c_{max} - s_2 - s_3)$ in steps of $step_s = 0.5$ms. Further, we compute the corresponding maximum end-to-end delay $\tau$ encountered by each controller and retain the feasible system configurations where the delay is *finite*.

We explored 273 system configurations and retained $|\Omega| = 15$ *feasible* configurations out of them. For example, with $c = 15$ms and $s_1 = 5$ms (and $s_2 = 3$ms; $s_3 = 2$ms), the delay encountered by the three controllers are (in ms): $\tau_1 = 80.0$; $\tau_2 = 99.0$; $\tau_3 = 113.0$.

In order to carry out the optimal co-design outlined in Section IV.D, we assumed that the plants corresponding to tasks $T_4, T_7$, and $T_{11}$ have a transfer function as in (15) with $K_p = 1000$. The corresponding controllers in tasks $T_3$, $T_6$ and $T_9$, $T_{10}$ were assumed to have a transfer function

$$C_j(s) = K_j(s + b_j)e^{-\tau_j s}, \quad j = 1, 2, 3.$$

| config. | $c$ [ms] | $s_1$ [ms] | $\lambda_k = \frac{1}{3}$, $k = 0, 1, 2$ | $\lambda_0 = \frac{1}{21}$, $\lambda_k = \frac{10}{21}$, $k = 1, 2$ |
|---|---|---|---|---|
| 1 | 10 | 3.5 | 1.5579 | **0.5662** |
| 2 | 10 | 4.0 | 1.5540 | 0.5671 |
| 3 | 10 | 4.5 | 1.5534 | 0.5675 |
| 4 | 10 | 5.0 | **1.5527** | 0.5664 |
| 5 | 15 | 5.0 | 1.5775 | 0.5913 |
| 6 | 15 | 5.5 | 1.5766 | 0.5908 |
| 7 | 15 | 6.0 | 1.5737 | 0.5882 |
| 8 | 15 | 6.5 | 1.5726 | 0.5883 |
| 9 | 15 | 7.0 | 1.5722 | 0.5885 |
| 10 | 15 | 7.5 | 1.5713 | 0.5877 |
| 11 | 15 | 8.0 | 1.5684 | 0.5869 |
| 12 | 15 | 8.5 | 1.5682 | 0.5867 |
| 13 | 15 | 9.0 | 1.5680 | 0.5864 |
| 14 | 15 | 9.5 | 1.5677 | 0.5859 |
| 15 | 15 | 10.0 | 1.5668 | 0.5846 |

TABLE I
COSTS $\tilde{\mathcal{P}}_i, i = 1, \ldots, 15$ AND OPTIMAL COSTS $\mathcal{P}^*$ (IN RED) FOR DIFFERENT $\lambda_k$'S

The delays $\tau_j$, $j = 1, 2, 3$ are determined using the *design space exploration* method outlined in Section IV.B. For each of the $i$ configurations 1 through 15, we index the corresponding delays as $\tau_{j_i}$, $i = 1, \ldots, 15$. The optimal co-design now consists of choosing $\mathcal{P}^*(\theta^*, i^*)$ defined in (27). Each of these $\tau_{j_i}$ is used as a parameter for (23) and (24). It was observed that for all positive values of $K$ and $b$, the system was stable. We therefore chose $\Gamma$ as the positive quadrant. Typical costs $\mathcal{P}^j_{k,i}$ are shown in Fig. 13 – 12, where $i = 4$ in Fig. 13 – 15 for $k = 0, 1, 2$, respectively, and $i = 10$ in Fig. 10 – 12, for $k = 0, 1, 2$, respectively. For ease of simplicity we focus on a subset of $\Gamma$ in all the figures where the performance is close to optimal. These figures show that the costs change as a function of the controller parameters $\theta$ and the configuration $i$ and also that the sensitivity of these costs depends on the costs themselves.

Computing $\tilde{\mathcal{P}}^j_i$ as given in (26) we get a range of costs for two different sets of weighting factors $\lambda_k = \frac{1}{3}$, $k = 0, 1, 2$ and $\lambda_0 = \frac{1}{21}$, $\lambda_k = \frac{10}{21}$, $k = 1, 2$, and is assembled in Table I. The former $\lambda$-set weights all costs equally, whereas the latter one emphasizes transient and steady-state performance.

| config. | $c[ms]$ | $s_1[ms]$ | $K^1$ | $b^1$ | $K^2$ | $b^2$ | $K^3$ | $b^3$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 3.5 | 0.0032 | 1 | 0.0034 | 1 | 0.0030 | 1 |
| 4 | 10 | 5.0 | 0.002 | 1 | 0.002 | 1 | 0.002 | 1 |

TABLE II
OPTIMAL VALUES FOR $c$, $s_1$, $K^{j*}$ AND $b^{j*}$, $j = 1, 2, 3$

Table I shows that configuration 4 is optimal for $\lambda_k = \frac{1}{3}$, $k = 0, 1, 2$ and that configuration 1 is optimal for $\lambda_0 = \frac{1}{21}$, $\lambda_k = \frac{10}{21}$, $k = 1, 2$ In Table II, the corresponding optimal values of the controller parameters $K$ and $b$ are shown.

## VI. CONCLUDING REMARKS

In this paper we proposed a scheme for optimizing the parameters of a hierarchical bus scheduler in order to improve various control performance metrics. We showed that the choice of values for these parameters heavily depend on the performance metrics used. As a part of future work, we plan to

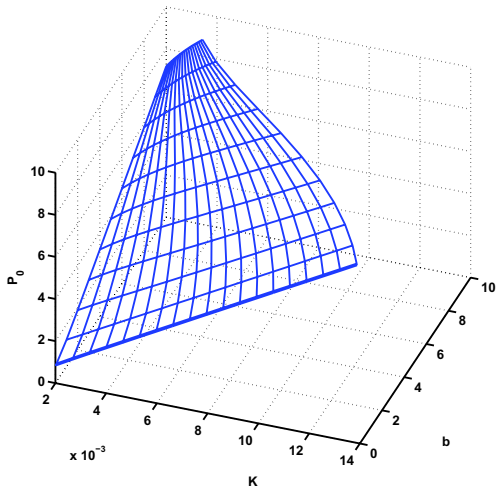


Fig. 10. The cost $\mathcal{P}^3_{0,10}$

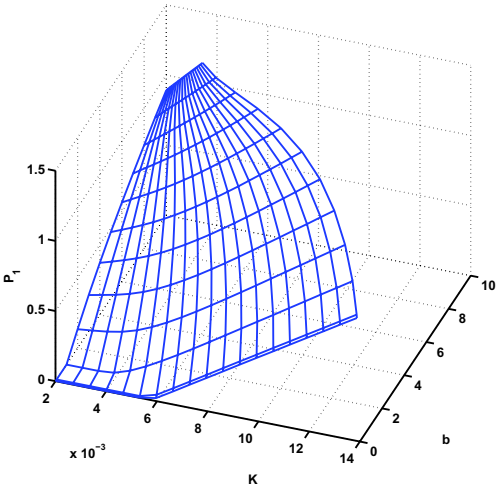


Fig. 11. The cost $\mathcal{P}^3_{1,10}$

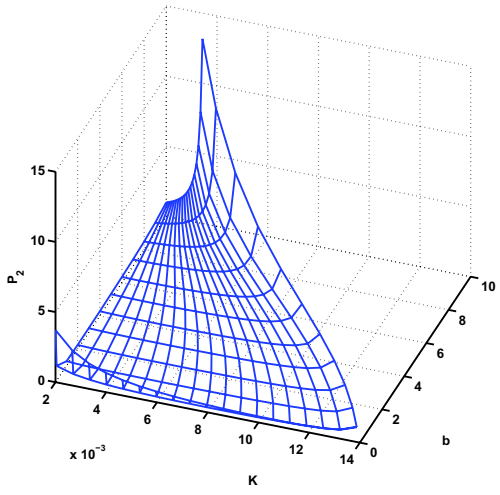


Fig. 12. The cost $\mathcal{P}^3_{2,10}$

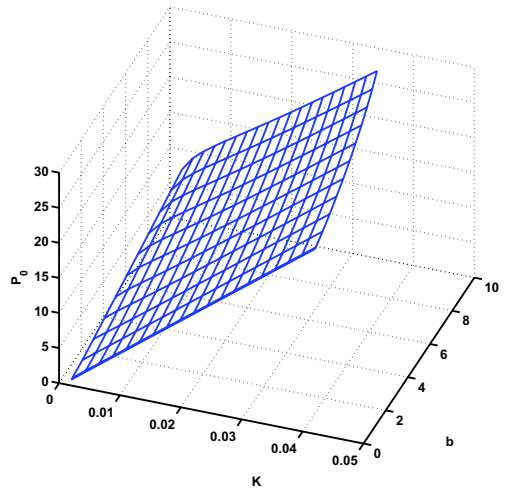


Fig. 13. The cost $\mathcal{P}^1_{0,4}$

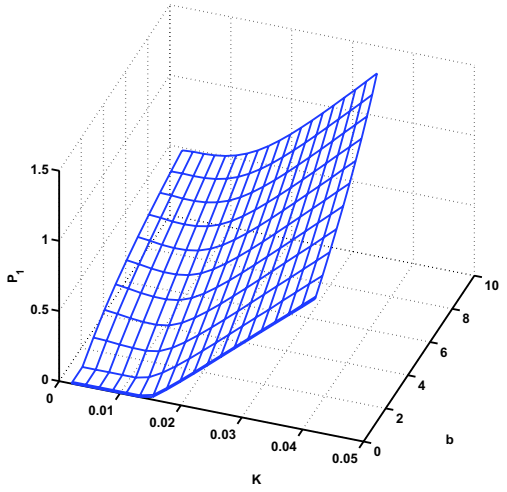


Fig. 14. The cost $\mathcal{P}^1_{1,4}$

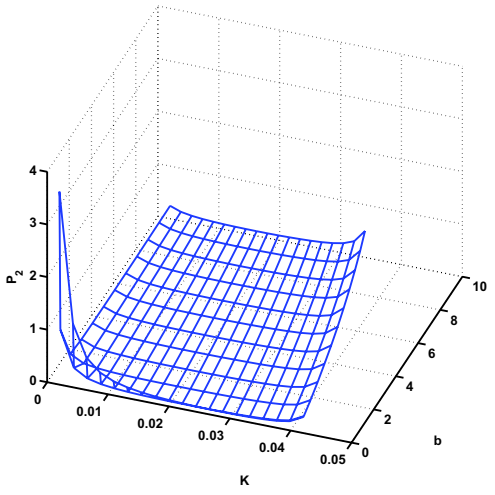


Fig. 15. The cost $\mathcal{P}^1_{2,4}$

extend these results to more general class of controllers (e.g., adaptive controllers).

## VII. Acknowledgement

This work was supported by the Technische Universität München - Institute for Advanced Study (TUM-IAS), funded by the German Excellence Initiative and by Deutsche Forschungsgemeinschaft (DFG) through the TUM International Graduate School of Science and Engineering (IGSSE). Parts of this work were carried out when the first author was a visiting student at the Laboratory for Information and Decision Systems (LIDS) at the Massachusetts Institute of Technology, and the fourth author was a Hans Fischer Senior Fellow at the TUM-IAS.

## References

[1] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard. Time-triggered implementations of dynamic controllers. In *Proc. 6th ACM & IEEE International conference on Embedded software (EMSOFT)*, 2006.

[2] H. Yazarel, A. Girard, G. J. Pappas, and R. Alur. Quantifying the gap between embedded control models and time-triggered implementations. In *Proc. 26th IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[3] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Proc. Design, Automation & Test in Europe (DATE)*, 2009.

[4] S. Samii, P. Eles, Z. Peng, and A. Cervin. Quality-driven synthesis of embedded multi-mode control systems. In *46th Design Automation Conference (DAC)*, 2009.

[5] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proc. 41st IEEE Conference on Decision and Control (CDC)*, 2002.

[6] A. Cervin and T. Henningsson. Scheduling of event-triggered controllers on a shared network. In *47th IEEE Conference on Decision and Control (CDC)*, 2008.

[7] E. Bini and A. Cervin. Delay-aware period assignment in control systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[8] L. Palopoli, C. Pinello, A. L. Sangiovanni-Vincentelli, L. Elghaoui, and A. Bicchi. Synthesis of robust control systems under resource constraints. In *5th International Workshop on Hybrid Systems: Computation and Control (HSCC)*, 2002.

[9] P. Naghshtabrizi and J. Hespanha. Analysis of distributed control systems with shared communication and computation resource. In *Proc. of the 2009 Amer. Contr. Conf.*, 2009.

[10] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[11] R. C. Dorf and R. H. Bishop. *Modern Control Systems*. Addison Wesley, 1995.

[12] A. Isidori. *Nonlinear Control Systems*. Springer, Berlin, 1995.

[13] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice Hall, 1990.

[14] I. M. Horowitz. *Synthesis of Feedback Systems*. Academic Press, New York, 1963.

[15] G. C. Goodwin, S. F. Graebe, and M. E. Salgado. *Control System Design*. Prentice Hall, 2001.

[16] Maria M. Seron, Julio H. Braslavsky, and Graham C. Goodwin. *Fundamental Limitations in Filtering and Control*. Springer, 1997.

[17] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[18] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[19] Ernesto Wandeler and Lothar Thiele. Optimal tdma time slot and cycle length allocation for hard real-time systems. In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 479–484, Piscataway, NJ, USA, 2006. IEEE Press.

[20] R.H. Middleton and J.H. Braslavsky. On the relationship between logarithmic sensitivity integrals and limiting optimal control problems. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 5, pages 4990–4995, 2000.