A Co-simulation Approach for System-Level Analysis of Embedded Control Systems

(Invited Paper)

Michael Glaß, Jürgen Teich, and Liyuan Zhang
Hardware/Software Co-Design, University of Erlangen-Nuremberg
Email: {glass, teich, liyuan.zhang}@cs.fau.de

Abstract—Control applications have become an integral part of modern networked embedded systems. However, there often exists a gap between control engineering and system design. The control engineer has detailed knowledge about the algorithms but is abstracting from the system architecture and implementation. On the other hand, the system designer aims at achieving high-quality implementations based on quality constraints specified by the control engineer. This may result in either an overdesigned system in case the specifications are pessimistic or an unsafe system behavior when specifications are too optimistic. Thus, future design automation approaches have to consider the quality of control applications both as design objectives and design constraints to achieve safe yet highly optimized system implementations. The work at hand introduces an automatic tool flow at the Electronic System Level (ESL) that enables the optimization of a system implementation with quality of control being introduced as a principal design objective, like the maximum braking distance, while respecting constraints like maximum slip to ensure maneuverability of a car. The gap between mathematically well-defined models for system synthesis and common analysis techniques for control quality is bridged by co-simulation: A SystemC-based virtual prototype of a distributed controller implementation is combined with high-level models of the plants specified in Matlab/Simulink. Through a model transformation, the traditional development process of control applications is combined with state-of-the-art ESL techniques, ensuring model consistency while enabling a high degree of automation.

I. INTRODUCTION

In modern means of transport like the automotive and avionics domain, many important control applications are implemented on heterogeneous distributed embedded systems that may consist of up to hundreds of Electronic Control Units (ECUs) as well as various sensors, actuators, and field bus systems. One important class of such applications are driver assistance systems in modern cars that, e.g., automatically keep a driver-specified speed of the car termed cruise control. In recent years, such applications have become more and more complex such that modern adaptive cruise control systems not only adapt the speed according to the driver’s setting, but also to the distance of cars running ahead and may even consider the surrounding of the car to predict whether the driver may overtake so to avoid needless braking. Another class are X-by-Wire applications that substitute mechanical and hydraulic systems for steering or braking. Here, the quality of the control applications is one of the key factors that determine their applicability with respect to (safety) constraints and the quality perceived by the consumer.

The design of such distributed embedded systems has become an extremely challenging task. Recently, Design Space Exploration (DSE) approaches at the Electronic System Level (ESL) have been developed, trying to assist the system designer in this task. The approaches aim at automatically investigating the tremendous design space and search for system implementations that are optimized with respect to several design objectives while meeting design constraints. Typical design objectives are, e.g., monetary costs, power consumption, or dependability. Design constraints are much more specific to applications and the system and may be, e.g., mounting space limitations, wiring capabilities, safety requirements, or the stability of certain control applications. From a system designer’s point of view, these objectives and constraints are typically predefined. Given these, DSE approaches assist the designer with an optimization loop that performs system synthesis and evaluation: During system synthesis, a candidate implementation of the system is derived by selecting architecture components, binding of tasks onto the components, routing of messages, and scheduling tasks and messages. The evaluation is responsible for analyzing each candidate implementation to quantify its design objectives and check whether all constraints are satisfied.

However, the quality of a control application may not be seamlessly translated into constraints like a maximum/average end-to-end latency or jitter. In fact, it is well known that, e.g., the distribution of end-to-end delays may have a significant impact on control quality [1]. In current practice, this existing gap may be bridged by control engineers specifying rather pessimistic design constraints. In such cases, the system implementation delivered by the system engineer may be in fact overdesigned which, of course, deteriorates design objectives. Avoiding such an overdesign, the control engineer may specify more optimistic design constraints that work fine for the average case. However, the system may not be able to deliver its correct service under all conditions, possibly resulting in low service quality of the application or even unsafe behavior.

To close this gap, the quality of a controller has to be considered directly during system design to avoid an inaccurate approximation based on design constraints only. This consideration is included in a controller synthesis with the discipline being often referred to as Control-Scheduling-Co-Design [2]. Existing approaches typically implement the controller itself as a set of periodic tasks that communicate via periodic messages, being mapped mostly to dedicated components. However, in modern distributed embedded systems, there is no dedicated subsystem per control application, but the tasks of different applications have to share computation as well as communication resources, causing varying end-to-end latencies or even message loss. To achieve good control performance, these effects that are a result of the mapping of control applications to a distributed system with shared media have to be taken into account. While several known approaches take these effects into account, they typically assume a fixed architecture platform and task mapping while taking into account the effects of different scheduling strategies [3], [4] only. When increasing the degree of freedom for the system designer by enabling variation in architecture, task mapping, routing, and scheduling, there exists a gap between the model
of the controller’s implementation and the required data for the control performance analysis, i.e., a model of the plant.

The work at hand presents a possibility to close this gap through co-simulation of an advanced system model including the controller and an advanced plant model. As pointed out in [5], classic system modeling languages like C are agnostic of the system behavior and may only cover functional behavior of a controller. As a remedy, the system model employed here comprises of an actor-oriented behavioral model that, combined with the current architecture, task mapping, message routing, and scheduling forms a virtual prototype of the implementation. The plant itself is modeled by the control engineer using a methodology that is well-established in his/her field like Matlab/Simulink [6]. Through co-simulation, a complete ESL design methodology is achieved that is capable of concurrently optimizing control quality as a principal design objective together with classic design objectives. Moreover, it enables to take design constraints like the stability of the control application or application-specific constraints like maximum braking distance into account. To further increase the degree of automation, it is outlined how to (semi-)automatically translate Matlab/Simulink models of sensors, actuators, and controllers into an actor-oriented System Description Language (SDL), in particular, SysteMoC [7]. This class of description languages is typically employed at the ESL since it provides executable, synthesizable, and often even verifiable system models. Particularly the aspect of translating a complete Matlab/Simulink controller model to SysteMoC, setting up a virtual prototype, and performing a co-simulation to determine control quality is presented using Brake-by-Wire with anti-lock braking system as a control application.

The rest of the paper is structured as follows: Section II introduces related work from the area of control-scheduling-co-design. Section III introduces ESL design fundamentals. The modeling of control applications in an ESL design flow is presented in Sec. IV. The proposed co-simulation approach to evaluate control quality is outlined in Sec. V. A Brake-by-Wire case study from the automotive domain is investigated in Sec. VI before the paper is concluded in Sec. VII.

II. RELATED WORK

In recent years, several design tools have been developed to help the designers to analyze and evaluate the control performance under timing influence. These tools may be coarsely divide into two groups, see [8] for a survey: Tools that focus on the statistical analysis of how timing affects control quality and tool flows that rely on system simulation.

Jitterbug is a widely used MATLAB-based tool that allows the computation of a quadratic performance criterion for a linear control system under various timing conditions [9]. The tool helps the designer to quickly evaluate how sensitive a control system is with respect to delay, jitter, lost samples, etc. Jitterbug is used in several works: In [10], the authors use Jitterbug to determine the cost function when considering the problem of optimal static period assignment for multiple independent control tasks executing on the same processor. In [11], the authors propose a control-scheduling co-design method that integrates controller design with both static and priority-based scheduling of the tasks and messages. The design objective is to optimize the overall control quality. In [3], [12], the authors present a methodology to consider delay distributions and not just worst case delays while optimizing the control performance. Limitations of Jitterbug include: (a) seamless applicability for linear systems only; (b) performance analysis is restricted to a single design objective, i.e., the quadratic cost; and (c) the delay distributions in each control application are assumed to be given and independent of each other. Particularly, the delay distributions cannot be derived from a behavioral model only since they are agnostic of system behavior, e.g., timing variation caused by contention on computation and communication resources shared between several control tasks.

TrueTime is a MATLAB/Simulink-based simulator for real-time control systems. It enables the co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics [13]. TrueTime uses kernel and network blocks to carry out the control tasks, which are characterized by a number of attributes like deadlines, release times, and priorities. Several existing works employ TrueTime to simulate and evaluate control systems, see [14], [15], [16], [17].

This work aims at overcoming the decoupling of delay distributions, considering multiple control quality objectives, and supporting ESL design for embedded control systems. The methodology proposed in the following applies co-simulation of a virtual prototype of the control application mapped to the system architecture and the high-level model of the physical environment. It is worth mentioning that recent works, see for example [18], aim at avoiding intensive control quality analysis by simulation or third-party tools and rely on formal models that efficiently approximate the control quality instead. This has the potential to significantly speed-up DSE.

III. ESL DESIGN FUNDAMENTALS

This work introduces the aspect of control quality into system design by developing adequate models for controllers and an evaluation of controllers and plants based on co-simulation. Thus, the methodology is capable of introducing classic control quality metrics and application-specific metrics as both principal design objectives and design constraints and into Electronic System Level (ESL) design. In this section, the required fundamentals in ESL design of embedded systems are outlined.

The vast majority of ESL design approaches, see [19] for a survey, is inspired by the Y-chart approach as schematically included in Fig. 1. The ESL design approach as employed here starts with an executable specification, i.e., a behavioral model of the functionality of the system. The models for available system components, cf. Model of Architecture (MoA) see [19], such as processors, buses, memory units, etc. are stored within a component library. From this, a graph-based exploration model is derived. The exploration model consists of an application that models the functionality and an architecture that models the available resources. During the phase of Design Space Exploration (DSE), a set of high-quality system-level implementations is delivered to the designer who chooses one (or several) to be refined in the next lower level of abstraction. The set of high-quality implementations results from the presence of multiple, often conflicting objectives that makes DSE a Multi-Objective Optimization Problem. During DSE, implementations are obtained by mapping the application onto the architecture in a process termed system synthesis. The quality of each implementation with respect to given objectives and its compliance with given design constraints is determined in an evaluation step.

A. Executable Specification

In [7], a library for modeling and simulating actor-oriented behavioral models termed SysteMoC is presented. SysteMoC is
Based on SystemC, a de facto standard for system-level modeling, adding actor-oriented Models of Computation (MoC) to develop an analyzable executable specification language. In actor-oriented models, **actors**, which encapsulate the system functionalities, are potentially executed concurrently and communicate over dedicated abstract **channels**. Thereby, actors produce and consume data (so-called **tokens**), which are transmitted by those channels. Actor-oriented models may be represented as bipartite graphs, consisting of channels and actors. An **actor** is a tuple \(a = (I,O,F,R)\) containing actor ports partitioned into a set of actor input ports \(I\) and a set of actor output ports \(O\), the set of functions \(F\) and the Finite State Machine (FSM) \(R\). The functions encapsulated in an actor are partitioned into **actions and guards** and are activated during transition of a the finite state machine (FSM) \(R\) that also represents the communication behavior of the actor (i.e., the number of tokens consumed and produced in each actor activation). An action produces outputs, which are used by the firing FSM to generate tokens for the FIFO channels connected to the actor output ports. A **guard** (e.g., `check` in Fig. 2) returns a Boolean value and the assignment of one or several guards to the FSM implements the required control flow. A **channel** is a tuple \(c = (I,O,n,d)\) containing channel ports partitioned into a set of channel input ports \(I\) and a set of channel output ports \(O\), its buffer size \(n \in N_\infty = \{1, \ldots, \infty\}\), and also a possibly empty sequence \(d \in D^*\) of initial tokens, where \(D^*\) denotes the set of all possible finite sequences of tokens. The basic SysteMoC model uses **FIFO** channels to present unidirectional point-to-point connection between an actor output port and an actor input port. Actors are only permitted to communicate with each other via channels, to which the actors are connected by ports. In a SysteMoC actor, the communication behavior is separated from its functionality, which is a collection of functions that can access data on channels via ports. A graphical representation of a simple SysteMoC actor is given in Fig. 2 which sorts input data according to its algebraic sign.

**B. Exploration Model**

For the DSE, an exploration model termed **specification** defines the available hardware components as well as the processing tasks that need to be distributed in the system. This graph-based specification (see Fig. 3(c)) consists of the platform **architecture**, the **application**, and a relation between these two views, the **mapping constraints**:

- The architecture is modeled by a graph \(g_a(R,E_a)\) and represents all available interconnected components, i.e., hardware resources. The vertices \(v \in R\) represent the resources, e.g., ECUs, buses, sensors or actuators. The edges \(E_a\) model available communication links between resources.
- The application is modeled by an application graph \(g_t(T \cup C,E_t)\) that represents the behavior of the system. The vertices \(t \in T\) denote **processing tasks** and the vertices \(c \in C\) **communication tasks**. The directed edges \(e \in E_t \subseteq (T \times C) \cup (C \times T)\) denote data dependencies between tasks, respectively actors.
- The relation between architecture and application is given by a set of **mapping edges** \(E_m\). Each mapping edge \(m \in E_m\) is a directed edge from a task to a resource, with a mapping \(m = (t,r) \in E_m\) indicating that a specific task \(t\) can be mapped to hardware resource \(r\).

**C. System Synthesis**

From the specification of the system that implicitly includes all design alternatives, a system level **implementation** has to be deduced, respectively synthesized. This implementation corresponds to the hardware/software system that will be implemented. The synthesis thereby involves the following steps:

- The **allocation** \(\alpha \subseteq R\) denotes the subset of the available resources that are actually used and implemented in the embedded system.
- The **binding** \(\beta \subseteq E_m\) is a subset of the mapping edges in which each processing task is **bound** to a hardware resource that executes this task at runtime.
- The **routing** \(\gamma \subseteq R\) of each communication task is a subset of resources over which a communication task is routed.
The schedule \( \phi \) can be either static (with predefined start times of the tasks) or dynamic (with assigned periods or deadlines to the tasks).

Thus, an implementation is given by a tuple \((\alpha, \beta, \gamma, \phi)\).

IV. Controller Modeling for ESL Design

Introducing a novel design objective (design constraint) requires (a) an adequate modeling and consideration during system synthesis and (b) providing an evaluation technique to quantify the design objective (to check for compliance with the design constraint). This section introduces the proposed system modeling approach. The employed control quality analysis technique based on co-simulation is presented in the next section. The overall modeling flow is depicted in Fig. 3.

A. Controller and Executable Specification

Control systems with feedback possess multiple advantages in comparison to open-loop systems. The structure of a quite general feedback control system is as follows, see Fig. 3(a): The current state of the system \( x \) is sampled by a sensor and a controller computes the new control signal which is sent to an actuator. The actuator generates the control vector \( u \) which affects the behavior of the physical system that is also called plant.

For the modeling of complex controllers and plants, control engineers typically employ rich tool boxes, Matlab/Simulink being one of the most prominent ones. The latter uses time-based block diagrams to present functional units that exchange data via signals (lines). Hence, there exists a significant similarity between Matlab/Simulink modeling and actor-oriented models as employed in ESL design that is the base of an automatic transformation pattern to derive an actor-based executable specification from Matlab/Simulink code: For every (non-hierarchical) block in Matlab/Simulink, a corresponding SysteMoC actor is stored in an actor library. Thus, every basic block that appears in a complex controller model can directly be transformed one-to-one to a SysteMoC actor. Each signal is (a) directly transformed to a SysteMoC channel with either FIFO or register semantics if it has only one sender and one receiver or (b) transformed to multiple channels with broadcasting if it has one sender and multiple receivers. In case the control engineer specified an S-function block, an empty actor stub with correct channels is created and has to be filled manually. Note that the detailed transformation procedure has to take into account different data types etc. and may have to be adapted to the respective SDL used.

B. Control Application Exploration Model

To take into account feedback control systems during DSE, the control applications are transformed to graphs \( g_i(T \cup C, E_i) \). In case of a multi feedback controller design problem, a system specification consists of a set \( P \) of plants. The application graph \( g_i \) of such a system is defined as:

\[
g_i = \bigcup_{p \in P} a_p
\]

(1)

Here, each plant \( p \) is controlled by a separate control application, modeled as a control application graph \( a_p \). Looking at a single control application, the control application graph \( a_p \) has to represent the control application of its state space model according to Fig. 3. Each control application graph \( a_p \) describes the part of the feedback control system that is subsequently implemented: the sensor sampling the state \( x_p \) of plant \( p \) and preprocessing this data; the controller computing the control vector; and the actuator generating the control signal \( u_p \). This leads to the following definition of a control application graph \( a_p \) of a feedback controller design problem for a plant \( p \):

- The set of sensor tasks \( S_p \) models the sensors monitoring plant \( p \). Each element \( s \in S_p \) is a vertex, denoting one single sensor task.
- The set of controller tasks \( H_p \), where each element \( h_p \in H_p \) is a vertex, denoting a processing task computing a part of the control vector.
- The set of actuator tasks \( U_p \), where each element \( u_p \in U_p \) is a vertex, denoting a processing task to generate one control signal for plant \( p \).
- The set of communication tasks \( C_p \), where each element \( c_p \in C_p \) is a vertex, represents a data transfer.
- The directed edges \( e_p \in E_p \subseteq (S_p \times H_p) \cup (H_p \times H_p) \cup (H_p \times U_p) \) model data dependencies between the tasks.

Thus, a control application graph \( a_p \) is defined as \( a_p((S_p \cup H_p \cup U_p \cup C_p), E_p) \). Note that this definition allows multiple sensor, controller, and actuator tasks that may be required to control a single plant.

The proposed exploration model may, in case the behavioral model can or shall not be derived as an intermediate step, be directly derived from the state-space model of the controller. If the executable specification is given as proposed here, the transformation from the executable specification to the respective exploration model can be performed fully automatically, following a transformation pattern presented in [20].

V. Control Quality Evaluation by Co-Simulation

In this section, a co-simulation approach for control quality evaluation is presented. First, the used performance estimation technique based on virtual prototyping of the system implementation is introduced. By co-simulating a Matlab/Simulink-based plant model and the virtual prototype of the system, i.e., sensors, controllers, and actuators mapped to architecture components, see [21], application-specific metrics such as the braking distance can be used to reflect control quality.

A. Performance Simulation by Virtual Prototyping

A key aspect of the evaluation phase addressed in this work is performance simulation. The goal is to evaluate the system behavior, i.e., the behavior of tasks and their communication when mapped to system components. The work at hand employs a SystemC-based performance simulation to form a Virtual Prototype VP of the system. In general, the VP consists of an application enriched with architecture information.

To execute a performance simulation for a VP, the concept of Virtual Processing Components (VPCs) [22] is used, a custom library for performance simulation of SystemC models. VPC has to be configured with the implementation determined by the system synthesis. A VPC model consists of three components: (1) the resources in the system, (2) the mapping of the tasks to the resources, and (3) the routing that specifies the communication paths. Each node in the architecture graph \( g_a \) represents a resource modeled in the VPC library. Resources include communication as well as computation resources. To allow execution of multiple tasks running on one VPC resource, a scheduling policy for each VPC resource is determined from the implementation. Each node from the application graph \( g_i \) is mapped to one VPC resource. This mapping is given by the implementation.

For a proper timing simulation, the computation delay is determined by the mapping. This delay depends on attributes
of the resources like the operations executed per second and of
the task like the number of instructions. The communication
between the tasks mapped to the resources has to be defined
also for VPC. For each communication task \( c \in C \), a route is
specified by hops across VPC resources. For each hop, some
parameters can be defined such as a message priority.

The virtual prototyping approach based on VPCs extends
the standard SystemC simulator; for the sake of comprehensive
introduction of the co-simulation, the complete VPC-based
virtual prototyping approach is termed SystemC simulator in
the following.

### B. Co-Simulation

Complex control applications often consist of multiple con-
trol loops that interact with each other to a large extend.
The kind of interaction may even be situation-dependent as
known from driver assistance systems that, e.g., may adapt to
changing environment. In these cases, the quality of a single
control loop can hardly capture the control quality of the
whole application or system. In fact, application-specific con-
trol quality metrics may become both comprehensive design
objectives such as the braking distance and valuable design
constraints such as the maximum slip of a wheel to ensure
maneuverability of a car. These metrics, however, vary from
application to application and their outcome may significantly
depend on the use case.

As outlined before, Matlab/Simulink offers a rich toolbox
to model complex and multiple plants but cannot consider
varying sensor to actuator delays. On the other hand, the
proposed performance estimation reflects the system behavior
of several complex controllers that interact with each other
but has serious drawbacks modeling complex plant behavior.

To derive the described application-specific control quality
metrics, a co-simulation approach is implemented in which the
control application including sensors and actuators is modeled
in SystemMoC while the plant is modeled in Matlab/Simulink.
The sensor actor in the VP reads the actual state \( x \) of the
plant periodically. The \( \text{sampling times} \) that are denoted as
\( \tau_1, \ldots, \tau_i, \ldots, \tau_n \) with \( \tau^* = \tau_{i+1} - \tau_i \) being the \( \text{sampling interval} \). After sampling the state \( x(\tau_i) \) at sampling time \( \tau_i \),
the implementation of the controller computes the control
vector \( u(\tau_i) \). The delay from sensor from controller to actuator
caused by computation, scheduling, and resource contention is
defined as \( d_i \). This delay \( d_i \) is determined during the discrete-
event-simulation of the VP. This means, after sampling the
sensor value at \( \tau_i \), the control vector \( u(\tau_i) \) is updated at \( \tau_i + d_i \). Hence, the plant has to be updated with respect to
the delay. Therefore, the control vector at an arbitrary time \( \tau \)
is given as follows:

\[
u(\tau) = \begin{cases} u(\tau_{i-1}), & \tau_i \leq \tau < \tau_i + d_i \\ u(\tau_i), & \tau_i + d_i \leq \tau < \tau_{i+1} \end{cases}
\]

Given both simulations run in parallel, synchronization mech-

1For FIFO-based channels in the actor-oriented model, the delay has to be always smaller than the sampling interval, i.e., \( \forall i : d_i < \tau^* \). For register-based channels, this assumption becomes obsolete. Just as in real-world controller implementations, the actuator may then skip some control vectors since these are updated and overwritten by a subsequent control vector.
of the co-simulation interface are (a) synchronizing the Matlab/Simulink simulator and SystemC simulator and (b) exchanging data between simulators via shared memory (or via sockets if these two simulators are not on the same computer).

Matlab/Simulink uses numerical techniques to simulate the behavior of continuous dynamic systems. It uses a configurable sampling rate to determine how often a Matlab/Simulink block is evaluated. During simulation, the output of a block is, hence, only updated at so-called simulation steps. In the co-simulation approach proposed here, the co-simulation interface has to control the size of simulation step, cf. also Fig. 5, to consider the sampling times and update of the control vector correctly. Although it seems that both simulators run in parallel, however, the actual co-simulation process is in parts serialized. The process of co-simulation for one sampling time $\tau_i$ is depicted in Fig. 5.

1) At each sampling time $\tau_i$, the Matlab/Simulink simulator computes the current state of a plant model $x(\tau_i)$, passes it to the co-simulation interface, and is then suspended.
2) The SystemC simulator is invoked by the co-simulation interface and reads $x(\tau_i)$.
3) The SystemC simulator computes the control vector $u(\tau_i)$.
4) The SystemC simulator sends $u(\tau_i)$ back to the co-simulation interface together with the calculated delay $d_i$.
5) The co-simulation interface invokes the Simulink simulator while at the same time, the SystemC simulator advances its simulation time to the beginning of the next sampling time $\tau_{i+1}$ and is suspended by the co-simulation interface.
6) The Matlab/Simulink simulator continues its simulation using the old control vector $u(\tau_{i-1})$ due to the delay $d_i$.
7) After the delay $d_i$ has passed, the co-simulation interface updates the control vector in the plant with the value of $u(\tau_i)$ by invoking the Matlab/Simulink simulation engine.
8) The Matlab/Simulink simulator continues its simulation with the updated control vector $u(\tau_i)$.

VI. CASE STUDY: BRAKE-BY-WIRE

As a case study, a Brake-by-Wire (BbW) system from the automotive domain shall serve as an example where application-specific control quality measures may serve as comprehensive design objectives and constraints. The complete system including both controller application (BbW), the plant, and the co-simulation interface, see Fig. 6, is modeled in Matlab/Simulink first. Then, the ABS braking controller is transformed into a SysteMoC behavioral model with the outlined transformation scheme. The BbW system contains several networked ECUs that communicate over a field bus to control the braking process on the four wheels, see Fig. 7. A simplified vehicle model based on the Quarter Car Model [24] is used here as the plant. The system behavior in terms of braking distance are considered as the application-based performance metrics in this test case and are measured during co-simulation of plant and controller. To highlight the behavior of the controller, the wheel rotational speed is observed as well, depicting when the ABS functionality comes into play. As a design constraint, the slip is investigated with a slip of 1 corresponding to sliding wheels and, hence, the requirement for a maneuverable vehicle is violated.

The upper and lower threshold for slip are set to 0.2 and 0.04 so the controller is applicable on icy road as well. The ABS controller monitors the deceleration of the four wheels. Right before a wheel locks up (slip = 1), the controller will release the brake. Because the inertia of the vehicle, wheel rotational speed raises up temporally. If the ABS controller detects the acceleration of the wheel, full brake starts again. The key is that the controller tries to keep the wheel slows
down nearly at the same rate as the vehicle. This process repeats multiple times until the vehicle stops.

The braking controller has been simulated with different architectures and sampling intervals, see Fig. 8 and Tab. I. For the CAN field bus with lower bandwidth (1 Mbit/s), the redundant main control actors are executed on a single ECU and an overall of five ECUs is used. The high bandwidth of the FlexRay bus (10 Mbit/s, single channel) enables to employ seven ECUs and higher sampling intervals, such that an improved control quality and dependability at increased cost may be expected. Note that not only the higher bandwidth, but the distributed mapping of the computation-intensive Main nodes shortens the end-to-end delay.

The co-simulation results of five implementations found during DSE are depicted in Fig. 9 and Fig. 10. Comparing VP1, VP2, and VP3, reducing sampling interval results in decreased braking distance and, hence, improved control quality but consumes more bandwidth. With respect to braking distance, see Fig. 10, VP4 and VP5 do not have a significant improvement in comparison to VP1-3. However, by using a redundant main computation node, the dependability and safety of the system is improved. Furthermore, investigating the slip, there is a change between VP4 and VP5, with the latter keeping the slip always in the specified optimal range, see Fig. 11. With the slip value in VP5 being always in the optimal range between 0.04 and 0.2, the ABS functionality is perfectly guaranteed according to the specification. Thus, the advantage of using a smaller sampling interval is still present, but it does not affect braking distance and whether the driver is affected or not may be subject to further investigations. However, this maybe not noticeable improvement is bought by consuming ten times more bandwidth (from 1.73% to 17.28%).

### VII. Conclusion

The quality of control in the sense of application-specific properties like braking distance or maneuverability of a car are becoming design objectives and constraints of utmost importance for future (distributed) embedded control systems. This must be considered by design automation approaches to achieve safe system implementations of high quality. This work presents a tool flow at the Electronic System Level (ESL) that enables the modeling, analysis, and optimization of a distributed controller systems with quality of control being considered as principal design objectives and constraints. The existing gap between actor-oriented models for system design and common analysis techniques for control quality is bridged by a co-simulation of a SystemC-based virtual prototype of the distributed controllers and plant models written in

### Table I

<table>
<thead>
<tr>
<th>#</th>
<th>bus</th>
<th>sampling interval $\tau^*$</th>
<th>bandwidth</th>
<th>delay $d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CAN</td>
<td>100 ms</td>
<td>1.02%</td>
<td>1.07 ms</td>
</tr>
<tr>
<td>2</td>
<td>CAN</td>
<td>50  ms</td>
<td>2.05%</td>
<td>1.07 ms</td>
</tr>
<tr>
<td>3</td>
<td>CAN</td>
<td>10  ms</td>
<td>10.24%</td>
<td>1.07 ms</td>
</tr>
<tr>
<td>4</td>
<td>FlexRay</td>
<td>10  ms</td>
<td>1.73%</td>
<td>0.499 ms</td>
</tr>
<tr>
<td>5</td>
<td>FlexRay</td>
<td>1  ms</td>
<td>17.28%</td>
<td>0.499 ms</td>
</tr>
</tbody>
</table>
Matlab/Simulink. This co-simulation not only allows to determine metrics from the control theory domain like quadratic cost or stability but also application-specific control quality metrics like the maximum braking distance. A presented model transformation combines the traditional development process of control applications with state-of-the-art ESL techniques, ensuring model consistency while enabling a high degree of automation. The design flow is implemented and tested here for a Brake-by-Wire control application from the automotive domain. The results of different architecture variants highlight the various trade-offs that exist between different design objectives as well as the fact that some improvements within the controller may not significantly influence principal application-specific design objectives like the braking distance.

REFERENCES


