Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment

Haibo Zeng, Marco Di Natale, *Member, IEEE*, Arkadeb Ghosal, and Alberto Sangiovanni-Vincentelli, *Fellow, IEEE*

***Abstract*—FlexRay is a new high-bandwidth communication protocol for the automotive domain, providing support for the transmission of time-critical periodic frames in a static segment and priority-based scheduling of event-triggered frames in a dynamic segment. The design of a system scheduling with communication over the FlexRay static segment is not an easy task because of protocol constraints and the demand for extensibility and flexibility. We study the problem of the ECU and FlexRay bus scheduling synthesis from the perspective of the application designer, interested in optimizing the scheduling subject to timing constraints with respect to latency- or extensibility-related metric functions. We provide solutions for a task and signal scheduling problem, including different task scheduling policies based on existing industry standards. The solutions are based on the Mixed-Integer Linear Programming optimization framework. We show the results of the application of the method to case studies consisting of an X-by-wire system on actual prototype vehicles.**

***Index Terms*—Automotive, FlexRay, mixed-integer linear programming (MILP), real-time distributed systems, scheduling.**

I. Introduction

THE development of new by-wire functions with stringent requirements for determinism and short latencies, and the upcoming active safety functions, characterized by large volumes of data traffic, generated by 360° sensors positioned around the vehicle, are among the motivations for the definition of the FlexRay standard. FlexRay is being developed by a consortium [15] that includes major car manufacturers such as BMW, Daimler-Benz, General Motors, Freescale, NXP, Bosch, and Volkswagen/Audi, as a new communication standard for highly deterministic and high-speed communication.

In FlexRay, the upper bound of communication speed is 10 mega bits per second (Mb/s), and the bandwidth is assigned according to a time-triggered pattern. Time is divided into communication cycles, and each cycle consists of four segments—*Static*, *Dynamic*, *Symbol Window*, and *Network Idle Time* [15]. Clock synchronization, embedded in the standard, ensures deterministic communication at no additional cost.

The *static* segment of the communication cycle enables the transmission of time-critical frames according to a periodic cycle, in which a time slot, of fixed length and in a given position in the cycle, is always reserved to the same node. The *dynamic* segment allows for flexible communications. Transmission of frames in the dynamic segment is arbitrated by identifier priority. FlexRay includes a dual channel bus specification and will include bus guardians for increased reliability and timing protection.

In the FlexRay static segment, each node only needs to know the time slots for its outgoing and incoming communications. The specification of these time slots is kept in local scheduling tables. No global description exists and each nodes executes with respect to its own (synchronized) clock. As long as the local tables are kept consistent, no timing conflicts or interferences arise. Slots that are left free in the (virtual) global table resulting from the composition of the local tables can be used for future extensions. Time protection and isolation from timing faults are guaranteed by the reservation of time slots and guardians that avoid node transmitting outside the allocated time window.

Given the tight time determinism in the communication scheduling, and the possibility of bus guardians, the static segment is suited for highly deterministic and safety-critical communication which could not be otherwise accommodated by the dynamic segment. As stated in [16] and [7], "the periodic and safety-critical data is scheduled on the static time-triggered segment while the dynamic segment is mainly used for maintenance and diagnosis data." In this paper, we focus on functions characterized by tight time determinism requirements and safety constraints–hence *our scheduling efforts are targeted only to the static segment*.

The communication cycle and the transmission slot time are design parameters that should be carefully selected. Fundamental issues related to the automotive supply chain: the composition of subsystems, future design extensibility, and reuse of components practically force standardization of these parameters for an automotive Original Equipment Manufacturer (OEM) and possibly across the supply chain. The selection of these parameters is discussed in the following. However, because of standardization, we will assume them as given inputs to our schedule synthesis.

Clock synchronization and time determinism on the communication channel allow the implementation of end-to-end computations in which the data generation, data consumption and communication processes are temporally aligned. This

Manuscript received October 04, 2009; revised April 04, 2010, July 24, 2010, and September 07, 2010; accepted October 04, 2010. Date of publication November 18, 2010; date of current version February 04, 2011. The work of A. Ghosal was done while he was a Senior Researcher at General Motors R&D. Paper no. TII-09-10-0250.

H. Zeng is with General Motors R&D, Palo Alto, CA 94306 USA (e-mail: haibo.zeng@gm.com).

M. Di Natale is with Scuola Superiore S. Anna, Via Moruzzi, 1, Pisa 56127, Italy (e-mail: marco@sssup.it).

A. Ghosal is with National Instruments Engineering Berkeley, Berkeley, CA 94704 USA (e-mail: arkadeb.ghosal@ni.com).

A. Sangiovanni-Vincentelli is with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: alberto@eecs.berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2010.2089465

1551-3203/$26.00 © 2010 IEEE

avoids sampling delays, and therefore removes the worst drawback of composable periodic activation semantics, that is lateness, in exchange of determinism. Also, system-level time-triggered schedules allow the semantics-preserving implementation of distributed control models (including models with synchronous reactive semantics, like those produced by popular commercial tools like Simulink from Mathworks [1]).

Time determinism requires that the time-triggered model of communication is propagated to the computation layers, using a time-triggered scheduler and a careful coordination of the communication and computation schedules. If the schedulers are not coordinated, then not only guaranteeing time determinism is more difficult and probably altogether impossible, but the performance of the system in terms of latency is significantly worse.

In this work, we explore two options, based on two existing standards for Real-Time Operating System (RTOS) and task scheduling: the time-triggered OSEKTime, and the fully preemptive, priority-based OSEK standard (OSEK stands for "Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen," that is: "Open Systems and their Interfaces for the Electronics in Motor Vehicles"). Both have been included (OSEK) or extended (OSEKTime) by the AUTOSAR (AUTomotive Open System ARchitecture) standard for automotive architectures and components [10]. Therefore, the results presented here also apply to the use of AUTOSAR-compliant operating systems.

However, AUTOSAR currently does not include explicit standardization support for schedule synchronization between the Electronic Control Unit (ECU) tasks and the FlexRay cyclic frames, but only defines communication by periodic sampling, where schedules are not synchronized. We discuss the limitations in the application of the AUTOSAR standard as is now and the applicability to our model. However, given the continuous evolution of AUTOSAR, it is very likely that an extension to a model with synchronized schedules will be produced in the future. Otherwise, it would be clearly inefficient to define time-triggered standards for ECU scheduling and communication networks, while leaving the interface between the two domains without synchronization.

The main focus of this work is the development of a methodology, based on an Mixed-Integer Linear Programming (MILP) formulation of the optimal scheduling of real-time tasks and communication signals mapped into FlexRay static segment frames.

II. Related Work

In automotive systems, computation and communication functions can be time- or event-triggered. In the first case, task activations and frame transmissions are bound to happen at predefined points in time. The Time-Triggered Protocol (TTP) [20] is one example. TTP uses a Generalized Time-Division Multiple-Access (GTDMA) scheme with variable sized slots, in which each node has only one opportunity to transmit in each cycle. In FlexRay, slots have the same size, but a node can have more than one transmission opportunity in each cycle.

In order to accommodate a fraction of traffic that is dynamically activated, flexibility can be added with an additional dedicated transmission window. This is the case of hybrid protocols like Byteflight [6], introduced by BMW for automotive applications and later superseded by FlexRay, and of Flexible Time-Triggered communication over Controller Area Network (FTT-CAN) [14]. The dynamic segment of FlexRay is similar to Byteflight and uses a priority-based arbitration for outgoing frames based on a virtual token concept.

In [27], the authors present an approach to timing analysis of applications communicating over a FlexRay bus. The authors first present a static cyclic scheduling technique for time-triggered messages transmitted in the static segment, which extends the previous work on TTP [13]. Then, they develop a worst-case response time analysis for event-based transmissions in the dynamic segment. Message analysis techniques are integrated in a holistic schedulability analysis algorithm that computes the worst-case response times of all the tasks and messages in the system. The analysis has been later improved in [28]. Finally, [17] presents a compositional analysis framework to compute worst-case end-to-end delays for FlexRay-based architectures. *All these works focus on the analysis side of the problem and consider the feasibility of a given solution with respect to deadlines*.

The design problem, that is, the synthesis of the FlexRay static communication schedule for the optimization of a design target is considered in [31] and [22]. In [22], both a fast heuristics and a complete MILP optimization formulation are proposed for the problem. The design target is the optimal schedule in terms of the number of used slots (minimizing the number of used slots). The paper assumes that the units of communication (to be mapped into FlexRay slots) are PDUs (Protocol Data Units). Therefore, compared to this work, the authors assume that communication signals are already packed into PDUs. In [31], in addition to the minimization of the number of used slots, the authors also present a formulation for the minimization of the transmission jitter. In this case, the method provides also the optimal packing of signals into frames (slots). The authors formulate the problem as a standard MILP and discuss reduction techniques. *Both works discuss optimization of the static segment communication schedule, but do not attempt optimization at the system-level. They do not consider possible end-to-end deadlines, information passing and precedence constraints among tasks and signals, nor synchronization of the task and signal schedules*.

In [16], the authors discuss a system-level design optimization problem. However, they assume a communication model where task and message schedules are not synchronized and the proposed solution is a heuristic, with no guarantee of optimality. Because of the asynchronous communication model, the problem considered in the paper is characterized by freshness constraints only (on the communication side). The optimization metrics are the minimization of the communication bandwidth (which of course only makes sense in an asynchronous task-to-message information passing model) and the minimization of the number of used slots.

Compliance with the AUTOSAR standard for message frame offset and periodicity definitions is included in the formulations in [22] and [16]. AUTOSAR constraints are also discussed in [21], with respect to their implications on the (end-to-end) response time analysis and network utilization bounds. In practice, the AUTOSAR FlexRay interface requires message periods to be strictly powers of 2 of the FlexRay cycle time. Concluding their study, the authors of [21] state that "This potential mismatch (between the true application periods and the AUTOSAR bounds) leads to a lower schedulable utilization for the AUTOSAR FlexRay interface . . . The least upper bound of schedulable utilization achievable under the AUTOSAR FlexRay interface is 50%, when the message-generating task periods are less than 128 communication cycles."

With respect to the cited works, our contributions can be summarized as follows.

- We present an optimization framework that includes not only signal to frame packing, but also frame to slot assignment (slot schedule), task schedule and the *synchronization of signal and task scheduling* considering end-to-end delays and the precedence constraints induced by information passing between tasks and signals.
- We provide an MILP formulation for this (NP-complete) problem, *instead of a heuristic method*. Formally, NP-completeness has been demonstrated for a subset of our problem, namely, the packing of signals (or more accurately, PDUs) into slots [22]. The possible advantages of an MILP solution are:
  - —*The availability of a bound for the cost of the optimal solution (which allows to evaluate the quality of the intermediate solutions provided by the solver) and* ***the possible guarantee of optimality in case the solver finds the optimum solution.***
  - —An MILP formulation is typically easier to retarget to a different optimization metric compared with heuristics; It also can easily accommodate additional constraints or legacy components which makes some design variables become constants in the formulation.
  - —For computing the solution it is possible to leverage availability of solvers that have been designed and programmed for good runtime and space performance.
  - —The solution is more easily formulated for reuse by any designer who wants to adopt it.
- Our MILP formulation includes the system-level schedule optimization with the definition of an *optimal relative phase* in the activation of tasks and signals which accounts for deadlines and precedence constraints.
- We provide reduction techniques for two metrics of interest. One, the *minimization of the number of used slots*, is probably the most popular metric for extensibility (and used in the greatest majority of the cited works), because any unassigned slot in FlexRay can be used by a new ECU connected to the network, and therefore matches the concept of extensibility at the ECU-level. The other metric is the maximization of the minimum laxity (difference between deadlines and response times) on paths selected by the designer.
- We show how the MILP formulation can actually compute the optimal solution for case studies taken from actual vehicles (not benchmarks or randomly generated system configurations) of industrial complexity. We provide experimental data (runtimes on systems of different size) to estimate the scalability of the proposed methods, where the problems are modeled in AMPL and solved using CPLEX 11.0 [11] on a machine with a 3.2 GHz CPU and 1 GB memory.

As for the possible limitations, we assume the FlexRay communication cycle, the static segment length and the slot size as predetermined. This assumption certainly allows to greatly simplify the problem complexity, but is mostly motivated and justified by the need of standardization, which is in turn a consequence of the automotive supply chain. Also, our work only partly accommodates current AUTOSAR recommendations. This is mostly caused by the lack of AUTOSAR support for systems where task and signal scheduling are synchronized. These limitations will be discussed at length in the following.

The organization of the rest of this paper is the following. Section III discusses two possible options for task and signal synchronization and information passing. Section IV provides an introduction to the system model used in this paper for tasks, signals and the FlexRay bus configurations. Section V describes the formulation of the schedule optimization for tasks and signals in MILP framework. The next two sections provide extended discussion of reduction techniques and experimental results with respect to [33]: Section VI presents the set of reduction techniques for the metric function of number of used slots, the results of the application to real systems case studies and a discussion on scalability; Section VII discusses another metric function of maximizing the minimum laxity among the paths. Finally, Section VIII concludes this paper.

III. Synchronization Modes

To leverage the full benefits of the FlexRay deterministic communication, it is necessary to synchronize the scheduling of tasks and signals. In this case, besides packing the signals into frames, designers will need to schedule the software tasks and frames, such that timing constraints including end-to-end deadlines are satisfied. In this work, we explore two options, based on the two existing standards for RTOS and task scheduling. One is the time-triggered OSEKTime standard. The other option is based on the fully preemptive, priority-based scheduling of the OSEK standard. There are two possible synchronization patterns between tasks and signals.

- **Asynchronous scheduling**. This model does not require that the job and signal schedulers are synchronized. Jobs (a job is defined as the instance of execution of a task) post data values for the output signals in shared variables. The communication drivers have the responsibility, at the beginning of each cycle or before each slot, to fill the registers for the outgoing communication slot. At the receiving side, the data are written into output registers and asynchronously read by the reader tasks.
- **Synchronized scheduling**. In this case, job executions and frame transmissions are synchronized in such a way that a job must complete before the beginning of the slot that transmits its output signal (with a margin determined by the necessary copy time). When schedulers are synchronized, it is possible to know what job produces the data that is transmitted by a frame and what is the job that reads the data delivered by the frame. Scheduling can be arranged to achieve very tight end-to-end latencies and small jitter

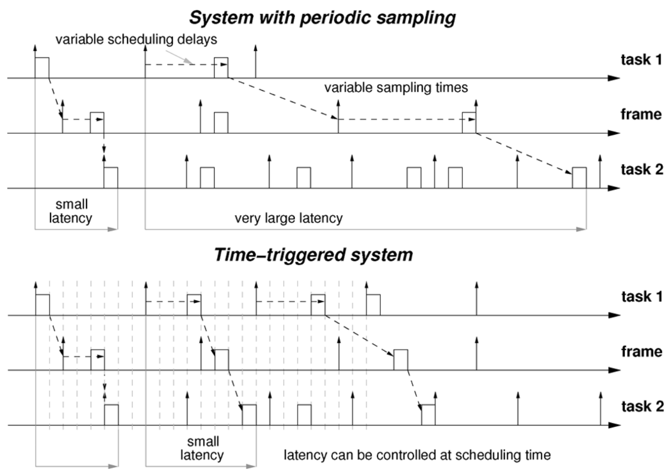


Fig. 1. Schedulers synchronized and not synchronized.

between the best and the worst-case response times. Also, equally important, this model can easily guarantee time-determinism and the preservation of the stream of data exchanged over the bus.

Fig. 1 shows two examples of scheduling without task and frame start and finish time synchronization and with synchronized instances, respectively. When schedulers are synchronized, sampling delays can be controlled and worst-case latencies can be reduced.

When considering the asynchronous scheduling, the FlexRay scheduling problem consists in the optimization of the communication scheduling for a set of periodic signal streams, generated at the ECU interface and considered independently from their sender and receiver tasks. For several car manufacturers this is a problem of high practical interest, because the first step in the move to FlexRay is likely to be the remapping of existing CAN communication flows, which are today typically asynchronous with respect to computations. This problem is addressed in [22].

Synchronization of Schedules and AUTOSAR Compliance: The current AUTOSAR specifications (4.0) [10] requires that the transmission of each signal is periodic. In an AUTOSAR flow, signals are mapped in frames (or PDUs, Protocol Data Units) and frames are then assigned to slots. Among the attributes that characterize each frame there is the assigned slot FRIF_SLOT_ID. Most importantly, AUTOSAR currently restricts the slots assignment to be periodic, with initial cycle FRIF_BASE_CYCLE and period FRIF_CYCLE_REPETITION, with the latter constrained to be a power of two ($\leq$64). This imposes strong requirements on the signal packing and the transmission periods of frames and signals, which makes almost impossible to construct a system in which schedulers are synchronized (except for the special cases in which the periods of the transmitter application tasks match exactly the allowed periods for frames).

The designer today has two options: be compliant with AUTOSAR and leverage all the advantages of the standard, included (but not limited to): improved portability, reusability, logging, debugging, and configuration. However, in this case, compliance with the rule comes at some price. Application signals must be constrained to be periodic with a period equal to a power of two of the FlexRay communication cycle time, or it is impossible to achieve synchronization between the scheduling of sender task, signal and receiver task, with a negative impact on end-to-end latency, a waste of bandwidth [21] and the need of additional synchronization layers [32] to recover the determinism that is necessary in safety-critical systems [30].

The second option is to synchronize schedulers and yield AUTOSAR compliance. Compliance with AUTOSAR is not strictly necessary for the development of FlexRay systems (and indeed several FlexRay systems are today developed without AUTOSAR support). In addition, AUTOSAR is in continuous evolution (timing support has only been added with release 4.0, approved in the very last months) and support for the synchronized scheduling of frames and tasks is likely to be provided in future versions.

Time aspects and better utilization, however, are probably not the main reasons why a synchronous model will be desirable or even necessary in the future. Many systems are today developed according to a model-based flow. Although AUTOSAR is agnostic with respect to SW development, very few people would disagree that it will have to be compatible with model-based development.

One of the tenets of model-based development is the capability of validating the models by simulation or formal reasoning before deployment of the target architecture. However, this implies that the translation of the models into software tasks and network messages preserves some semantics properties of the model (at least the ones verified by simulation or model checking). Currently used models (such as Simulink) are based on a deterministic synchronous reactive semantics. Translation of these models into an asynchronous platform, in which a random delay can occur at the interface between tasks and messages, may easily be the cause for the violation of the model behavior and unexpected results (this, of course, unless special synchronization layers [32] are deployed, with a further degradation of the performance).

The effect of delay on controls is formally discussed at length in [5]. Here, we only recall a very simple example from [25] showing what is the effect of a time delay at the interface between the output of a task and the transmission of a (FlexRay) message on the output of a model, as compared with the simulation output for zero delay. Fig. 2(a) shows a very simple Simulink system, in which two source blocks, a free counter modulo 16 and a repeating sequence generator producing the sequence [4, 2, 1], are executed with period 4 and feed a multiplier executing with period 1. We assume that the system is implemented in a distributed platform, where the data communications between the source blocks and the multiplier is scheduled over a FlexRay bus. Fig. 2(b) shows the simulation output with no delay at the interface between the computations and the communication link. Fig. 2(c) shows the same output, this time with some delay experienced by the signal upon transmission on the network. In the case of schedulers without synchronization, these delays appear in a pseudorandom fashion, changing the values of the output in a way that is difficult to predict, and typically spoils the result of any model-based verification.

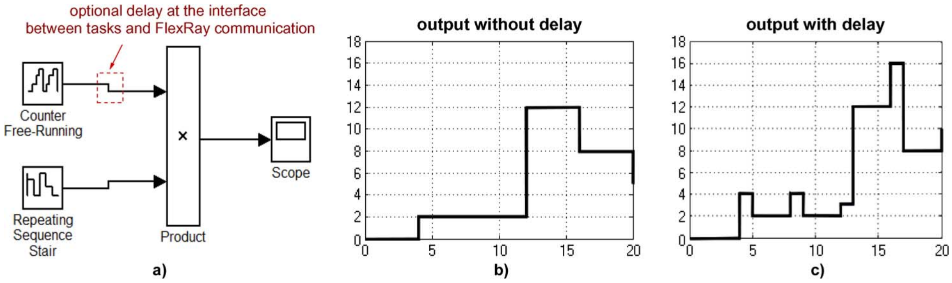


Fig. 2. Change in output because of (random) delay on communication.

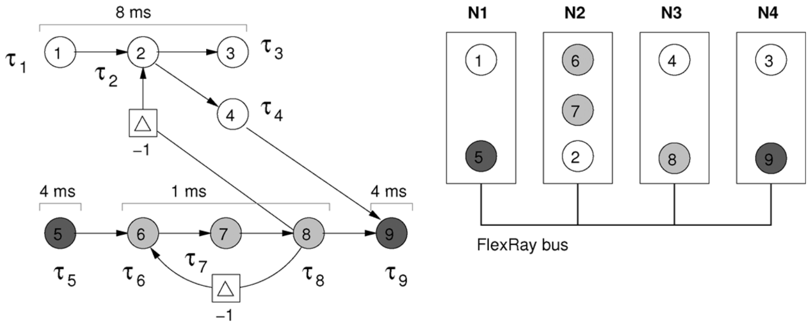


Fig. 3. System model with tasks, links, and delays.

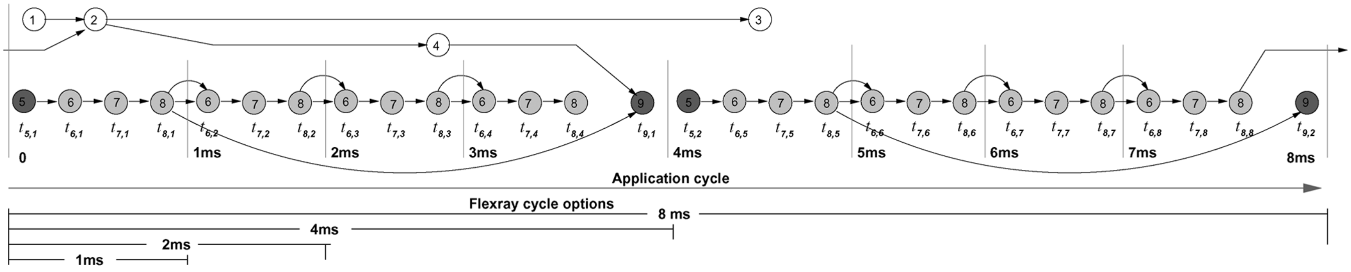


Fig. 4. Instance graph, application cycle, and FlexRay communication cycle.

Focus of the Paper: The focus of our paper is the *synchronization of signal and task scheduling*, considering end-to-end delays and the precedence constraints induced by information passing between tasks and signals. In this synchronization mode, there are two possible scheduling models for tasks in a FlexRay environment. In the first model, tasks are scheduled according to the OSEKTime framework. In OSEKTime, tasks are executed according to a time table, which defines their start times, and can optionally preempt each other. The other option is to schedule tasks in an OSEK framework in which tasks are periodically activated, each task has a fixed priority and the scheduler is preemptive with the option of preventing preemption inside selected task groups. In this paper, we provide a problem formalization and an MILP solution for both OSEK and OSEKTime task scheduling.

IV. Definitions, Notation, and Assumptions

We consider a model of the system computations as a *dataflow graph* V. The vertices represent the *tasks* and the edges represent the data *signals* communicated among them.

A task τ_i is characterized by the tuple $(e_i, T_i, \Phi_i, J_i, C_i, d_i)$, where e_i is the ECU resource it needs to execute, T_i is its period, Φ_i is its initial phase, J_i is its release jitter, C_i is its execution time, and $d_i \leq T_i$ is its deadline. For priority-based scheduling systems such as OSEK, each task τ_i is assigned a static priority P_i, and the scheduling policy is assumed to be fully preemptive. We use the convention in OSEK: the higher the number is, the lower the priority level is, thus $P_i < P_j$ implies that τ_i has a higher priority than τ_j. We also denote the set of tasks with higher priority than τ_i and executed on the same ECU as $hp(i) = \{j : P_j < P_i\}$.

An edge represents the input/output connections between tasks. An edge between tasks τ_h and τ_k denote a data signal $\sigma_{h,k}$ with a given bit width $b_{h,k}$ produced by τ_h and available to τ_k. For simplicity, signals will also be identified and denoted by a single index as in σ_i. Each periodic task reads its input at its activation time and writes its results at the end of its execution. Each signal may optionally be delivered with a *unit delay*, e.g., the signals from τ_8 to τ_2 and τ_6 in Fig. 3. Each signal also carries a precedence constraint in the execution of the sender and receiver job. If the signal is delivered without a unit delay, the successor must be executed after the sender job instance activated immediately before it, but before the following one; otherwise, the successor will use the signal value produced by the previous job of the sender (see Fig. 4).

Each task will run an infinite sequence of instances or *jobs*. The *application cycle* or *hyperperiod* H is defined as the least common multiple (*lcm*) of the periods of all tasks. Inside the hyperperiod, each job is considered as an individual scheduling entity and denoted as t_i. The scheduling problem consists of planning the execution of jobs and the transmission of signals into the available slots inside H. Jobs can also be denoted with reference to their task. In this case, $t_{k,j}$ denotes the jth job of task τ_k.

The *arrival time* of a job instance t_i is denoted as a_i or, using the instance index notation as $a_{k,j}$, with $a_{k,j} = \Phi_k + (j - 1) \times T_k$. It indicates the time instant when the job is signalled to be available for execution. The *release time* of a job is A_i, the time instant when the job is actually ready for execution. The *start of execution time* is s_i and the *finishing time* is f_i. The *response time* r_i of a job t_i is the time interval from its arrival to its termination, i.e., $r_i = f_i - a_i$. The worst-case task response time R_k of task τ_k is the maximum of the response times $r_{k,j}$ of its jobs $t_{k,j}$. The worst-case jitter J_k of task τ_k is the maximum difference between the arrival time and release time of all the jobs of τ_k (typically representing activation delays because of interrupt response times and the execution time of the scheduler itself).

The set of all the task instances transmitted in the application cycle defines the *application instance graph*, as in Fig. 4. Signals transmitted by tasks allocated to the same node may be transmitted in the data content of a frame m_i in a communication slot. We denote the start time of the jth slot inside FlexRay communication cycle k as $s^s_{k,j}$, and its finishing time $f^s_{k,j}$.

A *path* from τ_i to τ_j, or $P_{i,j}$, is a sequence $P = [\tau_i, \ldots, \tau_j]$ of tasks such that there is a link between any two consecutive tasks. For example, in Fig. 3, a path exists between tasks τ_1 and τ_9. The *latency of path* $P_{i,j}$ is defined as the time interval between the arrival of one instance of τ_i and the completion of the instance of τ_j that produces a result dependent on the output of τ_i.

The scheduling of FlexRay communication consists of the mapping of the tasks and signals defined in the application cycle into a set of communication cycle instances. This mapping can be performed in different ways, according to the selection of the communication cycle length, the size of the static segment, the slot size and correspondingly of the number of static slots in each communication cycle. It is practically impossible to encode all the above into an integrated problem formulation to be solved by an optimization framework. The resulting problem would very likely suffer from issues related to the size of the search space.

However, in practice, because of the structure of the supply chain and the need to reuse ECUs on different platforms, the fundamental parameters of the FlexRay communication cycle (FlexRay cycle length, size of the static segment, size of the static slot) are being standardized by most car integrators (automakers) and possibly across the industry. Otherwise, it would be impossible to reuse one component from a supplier in a different platform. Since suppliers work by numbers of hundreds of thousands and supply several automakers, there is a huge push for global standardization of these parameters. Each ECU is produced in large numbers (hundreds of thousands) and it would be difficult to change the communication schedule definition every time the ECU is used in a different platform. Also, ECUs can be integrated on the same bus only upon the condition that the communication parameters are defined in the same way and schedules do not conflict.

For sake of completeness, we summarize the outcome of the study in [31] (the only one we could find in our bibliography). The authors conclude that for the FlexRay communication cycle time it is favorable to use the largest possible value (power of two-divider of the application cycle). This largest value is very likely in contrast or even incompatible with the AUTOSAR requirements on frame periods. For the static slots size, the authors provide the set of constraints that bound the set of possible size definitions and then propose to try them all by enumeration.

In this work, we assume a FlexRay bus configuration $(H, l_{\text{comm}}, n_{\text{slot}}, l_{\text{slot}}, b_{\text{slot}})$ as given, where H is the length of the *application cycle* (the least common multiple of the task periods), l_{comm} is the length of the FlexRay communication cycle, n_{slot} is the number of slots in the static segment of the communication cycle, l_{slot} is the length of the slot in time, and b_{slot} is the size of the slot in bits. Based on this configuration, we apply a mathematical programming framework to encode the problem and synthesize other variables such as slot ownership, signal to slot mapping, frame and task scheduling.

We formulate our problem in the general framework of mathematical programming, where the system is represented with parameters, decision variables, and constraints over the parameters and decision variables. An objective function, defined over the same set of variables, characterizes the optimal solution. Our problem allows an MILP formulation that is amenable to automatic processing. An MILP program in standard form is

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of positive real or integer-valued decision variables. MILPs can be solved very efficiently by a variety of solvers. In this work, we make use of the CPLEX solver [11].

V. Problem Formulation

We use a mixed integer programming formulation to find a solution to the FlexRay scheduling problem with respect to a cost function that accounts for extensibility or timing performance.

A. Activation, Release and Deadline Constraints

Φ_i, T_i and d_i denote the initial phase, period and relative deadline for periodic task τ_i ($d_i \leq T_i$). a_i, A_i, s_i, f_i denote the arrival time, activation time, start time, and finish time for a job t_i. T_i and d_i are input parameters, while Φ_i, and consequently a_i, A_i, s_i (for OSEKTime scheduling only) and f_i are variables for the optimization framework. Given that all tasks are periodic with an initial phase, the arrival times of the jobs must be constrained accordingly

$$\forall t_{i,k}, a_{i,k} = \Phi_i + (k-1)T_i \tag{2}$$

$$0 \leq \Phi_i < T_i. \tag{3}$$

All jobs must finish before their deadlines.

$$\forall t_i, f_i \leq a_i + d_i \tag{4}$$

A_i is linked to a_i through J_i. a_i is the "ideal" periodic activation time, as provided by a hardware interrupt coming from a clock. A_i is the corresponding time when the periodic task (that should arrive at a_i) is released into the system, or more precisely, put into the ready queue by the interrupt handler routine. The difference between these two times is assumed as negligible for time-triggered OSEKTime systems where jobs are dispatched and executed according to a time table (there is no scheduler), and we assume dispatching delays to be negligible, hence $a_i = A_i$, however, this assumption is not true for OSEK systems, and we constitute the jitter J_i in our formulation. The job start times s_i must be constrained to be larger than the corresponding activation times

$$\forall t_i, A_i \leq s_i. \tag{5}$$

In OSEK, tasks are activated periodically, by an internal dispatcher or by an alarm and scheduled according to their priorities. As previously stated, the response time of the scheduler may introduce jitter in the activation time

$$\forall t_{i,k}, A_{i,k} - a_{i,k} \geq J_i. \tag{6}$$

The optimization variables Φ_i, a_i, s_i and f_i are all non-negative real numbers. In actual systems, the activation offset Φ_i is constrained to be a multiple of the system tick and should be more accurately modeled as an integer. However, this would greatly increase the complexity of the formulation with little advantage. In practice, we assume that the system tick is small enough to allow neglecting this quantization error (as in most cases). The final solution should however be checked for feasibility after the Φ_i have been quantized into the values allowed by the tick resolution on each ECU.

B. *OSEKTime Start Times and Preemption*

A binary variable is used to define the order of execution of jobs

$$y_{i,j} = \begin{cases} 0, & \text{if } s_i < s_j \\ 1, & \text{otherwise} \end{cases}. \tag{7}$$

The values of the y variables need to be kept consistent with the start times according to the definition

$$s_i < s_j + y_{i,j} \cdot M \tag{8}$$
$$s_j < s_i + (1 - y_{i,j})M \tag{9}$$

where M is a sufficiently large constant (larger than all other quantities involved in the formulas) This is the standard "big M" formulation of conditional constraints. Similarly, a binary variable is used to encode preemption between t_i and t_j on the same node with no data dependency

$$p_{i,j} = \begin{cases} 0, & \text{if job } t_i \text{ is not preempted by job } t_j \\ 1, & \text{otherwise} \end{cases}. \tag{10}$$

A set of constraints applies to the preemption variables. Clearly, mutual preemption is not allowed

$$p_{i,j} + p_{j,i} \leq 1. \tag{11}$$

If t_i starts later than t_j, t_j does not need to preempt t_i. This leads to an additional constraint between y and p

$$p_{i,j} \leq 1 - y_{i,j}. \tag{12}$$

The next inequality pair encodes additional properties of preemption with respect to starting and finishing times. If job t_i starts before t_j and is not preempted by t_j, then its finishing time should be less than or equal to the starting time of t_j. However, if t_i is preempted by t_j, then it needs to finish after the execution of t_j, as defined by the second constraint

$$f_i \leq s_j + y_{i,j} \cdot M + p_{i,j} \cdot M \tag{13}$$
$$f_j < f_i + y_{i,j} \cdot M + (1 - p_{i,j})M. \tag{14}$$

C. *Schedulability Constraints*

The schedulability constraints are modeled according to the rules for computing the starting, finishing times, and deadlines of all the jobs and signals (frames) scheduled or transmitted on the bus.

In OSEKTime, define θ as a set which contains all the job pairs (t_i, t_j), where t_i and t_j are mapped onto the same ECU without any data dependency (precedence constraints) between them and $[a_i, a_i + d_i] \cap [a_j, a_j + d_j] \neq \emptyset$. Equation (15) calculates the worst-case delay from the start to the finish of job t_i

$$f_i = s_i + C_i + \sum_{(i,j)\in\theta} p_{i,j} \cdot C_j. \tag{15}$$

In OSEK, the worst-case response time R_i is computed for the tasks and applies to all their jobs (where $hp(i)$ is the set of tasks with priority higher than τ_i)

$$R_i = J_i + C_i + \sum_{j\in hp(i)} \left\lceil \frac{R_i - J_i + J_j}{T_j} \right\rceil C_j. \tag{16}$$

Since all the parameters in (16) except R_i are known, R_i can be computed beforehand and used as a parameter. After optimization, data dependencies result in phase (offset) assignments for the activation of the tasks. Real-time schedulability theory tells us that the value of R_i computed according to (16) is the largest possible for any possible offset assignment and it is therefore safe to use (but pessimistic) for the purpose of defining end-to-end deadline constraints and a latency metric. The finishing time of OSEK tasks is computed as follows:

$$f_i = a_i + R_i. \tag{17}$$

D. *FlexRay Protocol Rules*

In a communication cycle, $s^s_{j,k}$ is used as an input parameter which denotes the starting time of the k^{th} slot from the

j^{th} communication cycle. $s^s_{j,k}$ is easily calculated as $s^s_{j,k} = j \cdot l_{\text{comm}} + k \cdot l_{\text{slot}}$.

The mapping of signals to slots is encoded in a set of binary variables

$$A_{i,j,k} = \begin{cases} 1, & \text{if } \sigma_i \text{ is mapped to cycle } j, \text{ slot } k \\ 0, & \text{otherwise} \end{cases}. \tag{18}$$

If a signal σ_i is mapped to the k^{th} slot of the j^{th} cycle, the start time s_i and finish time f_i of σ_i are automatically constrained to the start time $s^s_{j,k}$ and finish time $f^s_{j,k}$ of the slot

$$s^s_{j,k} \le s_i + (1 - A_{i,j,k})M \tag{19}$$

$$s_i \le s^s_{j,k} + (1 - A_{i,j,k})M \tag{20}$$

$$f_i = s_i + l_{\text{slot}}. \tag{21}$$

Each signal can only be mapped to one slot and the sum of the payloads over all the signals mapped into a specific slot will be upper bounded by the slot size

$$\sum_{s^s_{j,k} \le t_{\max}} A_{i,j,k} = 1 \tag{22}$$

$$\sum_{i \in \text{Signals}} A_{i,j,k} \cdot b_i \le b_{\text{slot}} \tag{23}$$

where $t_{\max}$ is the maximum time span over which the planner must compute a schedule (see Section V-G).

Each slot is owned by an ECU or it is free. A set of binary variable encodes the status of each slot

$$A_{e_i,j} = \begin{cases} 1, & \text{if slot } j \text{ is owned by ECU } e_i \\ 0, & \text{otherwise} \end{cases}. \tag{24}$$

FlexRay has its requirement for the slot ownership. If a slot is owned by one specific ECU, then the ownership applies to every communication cycle. Constraint (25) encodes slot ownership. If signal σ_i is mapped to communication cycle j and slot k, then its source ECU must own slot k. (26) ensures that every slot is owned by at most one ECU. If no signal is mapped to slot k in any communication cycle, then constraint (27) sets the slot ownership to null

$$A_{i,j,k} \le A_{e_i,k} \tag{25}$$

$$\sum_{e_p \in \text{ECUs}} A_{e_p,k} \le 1 \tag{26}$$

$$A_{e_p,k} \le \sum_{i \in \text{signals}, j < n_a} A_{i,j,k} \tag{27}$$

where n_a is the number of communication cycles in $[0, t_{\max}]$.

E. Data Dependencies

If there is a data dependency between two jobs or between a job and a signal, then all successors must start later than the predecessors. For example, if a job t_i sends a signal σ_j to some other job t_l, then we need to make sure that t_i finishes execution before the signal is scheduled for transmission on the bus, and the signal finishes its transmission before the receiver job t_l arrives. The following constraints encode these requirements, where $\gamma_{i,j}$ represents the worst-case copy time for the outputs to be written into the data transmit register in the FlexRay adapter, and $\gamma_{j,l}$ the copy time into the input variable for the receiving job:

$$f_i \le s_j - \gamma_{i,j} \tag{28}$$

$$f_j \le a_l - \gamma_{j,l}. \tag{29}$$

If the communication between jobs t_i and t_l occurs on the same node, a similar condition needs to be defined to ensure that the sender finishes execution before the arrival of the receiver in case they are executed by an OSEKTime system

$$f_i \le a_l. \tag{30}$$

However, (30) is over-constrained for OSEK tasks which are scheduled according to their priorities. If the sender job t_i has a higher priority than t_l, then we only need to guarantee that t_i arrives before t_l. Thus, (30) can be refined as follows:

$$\begin{cases} f_i \le a_l, & \text{if } P_i < P_l \\ a_i \le a_l, & \text{otherwise} \end{cases}. \tag{31}$$

Also, to avoid that the data in signal σ_j is overwritten by the next job (t_{i+1}) of the same sender task, σ_j needs to start transmission before the arrival of t_{i+1}, i.e.,

$$s_j \le a_{i+1}. \tag{32}$$

F. Legacy Components

For legacy components, the scheduling of tasks and signals is predefined, thus the values of design variables such as the task phase and signal to slot mapping (18) are given.

G. Scheduling Window

If Φ_i is the initial phase of a generic task τ_i, the scheduling of the tasks and of the FlexRay bus must be performed until an entire application cycle has been computed. This means that the schedule must continue until time $H + \max_i(\Phi_i)$. Since the initial phase values are computed as a result of the optimization, we will use an upper bound for the previous formula

$$t_{\max} = H + \max_i(T_i). \tag{33}$$

In the interval $[0, t_{\max}]$ (see Fig. 5), we need to schedule for each task τ_i a number of instances

$$n_i = \left\lceil \frac{t_{\max}}{T_i} \right\rceil. \tag{34}$$

However, not all of those instances can be scheduled freely. In the example of Fig. 5, this is true for $t_{3,3}$, but not for $t_{1,7}$, which must be scheduled in a position defined by $t_{1,1}$ because they are actually the same instance in the hyperperiod cycle. This translates into the constraint on the finish times for both types of schedulers. This translates into the constraint on the finish times for both types of schedulers

$$f_{i,q} = f_{i,k} + H \text{ where } q = nc_i + k. \tag{35}$$

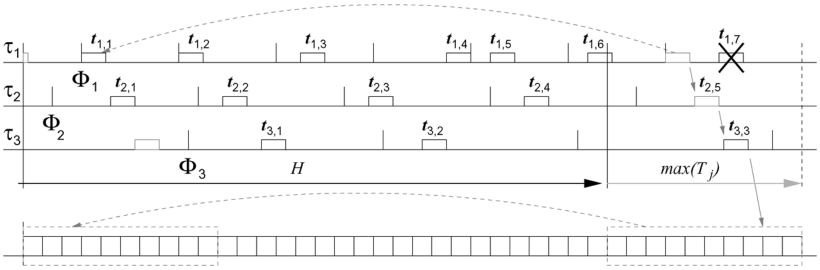


Fig. 5. Extent and constraints in the definition of the scheduling domain.

For a time-triggered (OSEKTime) scheduler, it must also be

$$s_{i,q} = s_{i,k} + H \text{ where } q = nc_i + k \tag{36}$$

nc_i is the number of jobs in one hyperperiod for task τ_i

$$nc_i = \frac{H}{T_i}. \tag{37}$$

Similar constraints exist on the scheduling of the FlexRay slots. We need to schedule beyond the application cycle, up to n_a number of cycles, where

$$n_a = \left\lceil \frac{t_{\max}}{l_{\text{comm}}} \right\rceil \tag{38}$$

in the last cycle, however, only

$$n_l = \left\lfloor \frac{t_{\max} - (n_a - 1)l_{\text{comm}}}{l_{\text{slot}}} \right\rfloor \tag{39}$$

slots need to be considered. Similar to job scheduling, signal to slot mappings must match when they refer to the same position in the application cycle. The matching set of signals can be identified as follows. If the sender and receiver jobs of signals σ_i and σ_j are exactly one application cycle away, i.e.,

$$\begin{aligned} &\text{src}(\sigma_i) = t_{k,l} \wedge \text{dst}(\sigma_i) = t_{m,n} \\ \wedge\ &\text{src}(\sigma_j) = t_{k,p} \wedge \text{dst}(\sigma_j) = t_{m,q} \\ &\text{where } p = l + nc_k \text{ and } q = n + nc_m \end{aligned} \tag{40}$$

then the two signals are actually the same and must be allocated to the corresponding slots (with a distance of H). We denote this relationship as $\sigma_i = \sigma_j$. For each cycle k and slot index l and, for each pair $\sigma_i = \sigma_j$, it must be

$$A_{j,m,l} = 1 \text{ if and only if } A_{i,k,l} = 1 \wedge m = k + H/l_{\text{comm}}. \tag{41}$$

H. AUTOSAR Compliance

As previously stated, AUTOSAR compliance requires each signal to be periodic, that the mapping of signals into frames or PDUs is statically defined. Similarly, PDUs are statically assigned to slots. AUTOSAR also requires the slots assignment to be periodic, with initial cycle FRIF_BASE_CYCLE and period FRIF_CYCLE_REPETITION, with the latter constrained to be a power of two ($\leq$64).

With respect to our formulation, when periodicity is enforced, it may actually be beneficial and used to further restrict the search space. In the model in which task and signal schedules are synchronized, AUTOSAR requirements can only be met if the period of all the sender tasks is a power of 2 multiple of the communication cycle. In this case, the formulation can be extended to account for the required periodicity of signals by adding a set of constraints. *However, in the following experiments, we do not enforce such a constraint as the FlexRay system is developed without AUTOSAR support.*

I. Objective Functions

Subject to the satisfaction of deadline constraints, we can seek optimality with respect to different cost functions. A very important objective, related to extensibility, is to minimize the number of used slots

$$\text{minimize} \sum_{e_i \in \text{ECUs}, j < n_{\text{slot}}} A_{e_i,j}. \tag{42}$$

If $\mathbb{P}$ is the set of latency-sensitive paths, we can alternatively minimize the sum of the end-to-end delays

$$\text{minimize} \sum_{p \in \mathbb{P}} f_{\text{snk}(p)} - a_{\text{src}(p)} \tag{43}$$

where $\text{src}(p)$ and $\text{snk}(p)$ are the source and sink object of the path p respectively. We can also maximize the minimum laxity (difference between the deadline of path D_p, and the finish time of the sink object) among all these paths

$$\text{maximize} \sum_{p \in \mathbb{P}} D_p + a_{\text{src}(p)} - f_{\text{snk}(p)}. \tag{44}$$

VI. Experimental Results: Minimization of Used Slots

A. Case Study: An Automotive X-by-Wire System

The application configuration of Tables I and II shows the data of a prototypical X-by-Wire application from a major automotive OEM. The application has 10 ECUs interconnected by a single FlexRay bus. There are a total of 47 tasks, with periods of 1 and 8 ms, respectively, and 132 signals. Tables I and II show periods and worst-case execution time of tasks, in microseconds and the size of each signal, in bits.

The problem is modeled in AMPL and solved using CPLEX 11.0 with a time limit of one hour, on a machine with an Intel Pentium 3.2 GHz CPU and 1 GB memory.

In this section, we discuss the reduction techniques and experiment results when the objective is to find the schedule with the minimum number of used slots. Optimization of the number of used slots is indeed a metric dictated by practical concerns

TABLE I
TASKS FOR THE X-BY-WIRE EXAMPLE

Task	ECU	Period	C_i	Task	ECU	Period	C_i
τ_8	e_9	8000	810	τ_{21}	e_5	1000	25
τ_9	e_9	8000	550	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	$e_5/e_6/e_7/e_8$	1000	60
τ_{11}	e_9	8000	100	$\tau_{23}/\tau_{27}/\tau_{31}/\tau_{35}$	$e_5/e_6/e_7/e_8$	1000	40
τ_{12}	e_9	8000	770	$\tau_{24}/\tau_{28}/\tau_{32}/\tau_{36}$	$e_5/e_6/e_7/e_8$	1000	20
τ_{13}	e_9	8000	200	$\tau_{25}/\tau_{29}/\tau_{33}$	$e_6/e_7/e_8$	1000	30
τ_{14}	e_9	8000	110	$\tau_{37}/\tau_{42}/\tau_{47}/\tau_{52}$	$e_1/e_2/e_3/e_4$	8000	1000
τ_{15}	e_9	8000	550	$\tau_{38}/\tau_{43}/\tau_{48}/\tau_{53}$	$e_1/e_2/e_3/e_4$	8000	500
τ_{16}	e_{10}	8000	780	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$	$e_1/e_2/e_3/e_4$	8000	1500
τ_{10}	e_{10}	8000	510	$\tau_{40}/\tau_{45}/\tau_{50}/\tau_{55}$	$e_1/e_2/e_3/e_4$	8000	1300
τ_{17}	e_{10}	8000	190	$\tau_{41}/\tau_{46}/\tau_{51}/\tau_{56}$	$e_1/e_2/e_3/e_4$	8000	350
τ_{18}	e_{10}	8000	260	τ_{20}	e_{10}	8000	230
τ_{19}	e_{10}	8000	100				

TABLE II
SIGNAL LIST FOR THE EXAMPLE

Signal	Send	Size	Recv	Signal	Send	Size	Recv
σ_1 to σ_4	τ_{15}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{48}	τ_{22}	32	τ_{16}
σ_5 to σ_8	τ_{20}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{49}	τ_{26}	32	τ_{16}
σ_9 to σ_{11}	τ_{23}	32	$\tau_{27}/\tau_{31}/\tau_{35}$	σ_{50} to σ_{53}	τ_{26}	16	τ_8/τ_{16}
σ_{12}, σ_{13}	τ_{21}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{54} to σ_{63}	τ_{39}	16	τ_{12}
σ_{14}	τ_{12}	8	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$	σ_{64}, σ_{65}	τ_{42}	16	τ_{12}
σ_{15} to σ_{18}	τ_{22}	16	τ_8/τ_{16}	σ_{66} to σ_{77}	τ_{44}	16	τ_{12}
σ_{19}, σ_{20}	τ_{23}	8	τ_8/τ_{16}	σ_{78} to σ_{87}	τ_{49}	16	τ_{12}
σ_{21} to σ_{23}	τ_{27}	32	$\tau_{23}/\tau_{31}/\tau_{35}$	σ_{88}, σ_{89}	τ_{52}	16	τ_{12}
σ_{24}, σ_{25}	τ_{25}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{90} to σ_{99}	τ_{54}	16	τ_{12}
σ_{26} to σ_{32}	τ_{12}	16	$\tau_{39}/\tau_{44}/\tau_{49}/\tau_{54}$	σ_{100}	τ_8	1	τ_{17}
σ_{33}, σ_{34}	τ_{27}	8	τ_8/τ_{16}	σ_{101} to σ_{116}	τ_{12}	16	τ_{17}
σ_{35} to σ_{37}	τ_{31}	32	$\tau_{23}/\tau_{27}/\tau_{35}$	σ_{117} to σ_{124}	τ_{12}	16	τ_{17}/τ_{18}
σ_{38}, σ_{39}	τ_{29}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{125}	τ_{30}	1	τ_8/τ_{16}
σ_{40}, σ_{41}	τ_{30}	16	τ_8/τ_{16}	σ_{126} to σ_{128}	τ_{34}	16	τ_8/τ_{16}
σ_{42} to σ_{44}	τ_{35}	32	$\tau_{23}/\tau_{27}/\tau_{31}$	σ_{129}, σ_{130}	τ_{37}	16	τ_{12}
σ_{45} to σ_{47}	τ_{33}	32	$\tau_{22}/\tau_{26}/\tau_{30}/\tau_{34}$	σ_{131}, σ_{132}	τ_{35}	8	τ_8/τ_{16}

and by the supply chain. Having more free slots means the possibility of accommodating an additional ECU, which is an option that far outweighs other extensibility metrics. Incidentally, this is also the metric used in most referenced works. We tried two FlexRay bus configurations. In both cases, the application cycle is $H = 8$ms. For the first configuration (referred to as *bus configuration 1*), the communication cycle $l_{\text{comm}} = 1$ ms with $n_{\text{slot}} = 22$, and the slot size $l_{\text{slot}} = 200$ bits, or 35 μs. In the second configuration (*bus configuration 2*) it is $l_{\text{comm}} = H = 8$ ms. The slot size is kept unchanged and there are $n_{\text{slot}} = 222$ slots in the cycle.

If the problem formulation in Section V is applied "as is" to the case studies, the solver can find a feasible solution, but not the optimal one within the time limit [Method (1) in Table III]. For example, for the OSEK scheduling with bus configuration 1 (top left of Table III), there are 32672 binary variables after the AMPL presolve phase. CPLEX finds a feasible solution with 13 assigned slots, but cannot guarantee optimality.

B. Restricting the Search Space

MILP problems generally require a significant amount of computer time and memory, and often the user needs to provide additional knowledge on the problem to further restrict the search space. We discuss several techniques to help solving our problem. Some of the techniques preserve the optimality of the solution, while the others over-constrain the problem and can settle for lower quality solutions. In the latter case, a lower bound on the cost of the optimal solution can be obtained and compared with the obtained solution.

1) Bounding the Number of Slots Required by Each ECU: If the objective is to minimize the number of used slots, it is possible to leverage the definition of a (tight) lower bound on the number of slots $k_i = \sum_k A_{e_i,k}$ that are needed by each ECU e_i. This bound can be obtained by solving a set of child problems F_{e_i}, one for each ECU e_i, in which we only consider the subset of jobs and signals on e_i and the constraints related to them, along with all the jobs that receive signals from e_i. Each such problem is a (much smaller) subset of our original problem F. Its solution $k_i = \text{Optimal}(F_{e_i}) = \min(\sum_k A_{e_i,k})$ provides a lower bound on the number of slots required by e_i in the original problem. Therefore, we can add to the original problem formulation a set of constraints

$$\sum A_{e_i,k} \geq k_i \tag{45}$$

which restricts the search space. If a solution with objective function equal to the lower bound is found, the optimization procedure can terminate immediately.

In case the problem size is still too large to find the optimal solution in a reasonable amount of time, we can settle for a suboptimal solution. Iteratively, we run the solver for some time and then take the best solution found. The signal to frame packing is fixed for the ECUs that reached their lower bound on the number of used slots, and the optimization procedure is repeated for the signal to slot assignments for the other ECUs. The iteration stops

TABLE III
RESULTS ON THE X-BY-WIRE EXAMPLE

		OSEK+bus confg1					OSEK+bus confg2				
Method	**Iter**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**
(1)		32672	88645	3600	13	No	42916	112757	3600	44	No
(1)+(2)	1	32672	88656	431.0	13	Yes	42916	112768	3600	46	No
	2	-	-	-	-	-	2404	7397	1305.2	44	Yes
(1)+(2)+(3)	1	25886	77578	365.7	13	Yes	34412	97620	2517.4	44	Yes
(1)+(2)+(3)+(4)	1	25886	76913	328.0	13	Yes	34412	97002	2099.7	44	Yes

		OSEKTime+bus confg1					OSEKTime+bus confg2				
Method	**Iter**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**
(1)		47878	131406	3600	13	No	62472	166519	3600	45	No
(1)+(2)	1	47878	131417	291.3	13	Yes	62472	166530	3600	45	No
	2	-	-	-	-	-	4344	15374	9.7	44	Yes
(1)+(2)+(3)	1	33380	102031	321.0	13	Yes	43646	126403	3600	46	Yes
	2	-	-	-	-	-	7012	25413	5.5	44	Yes
(1)+(2)+(3)+(4)	1	32048	96652	564.2	13	Yes	42314	121012	3018.4	44	Yes

		OSEKTime+Preemp+bus confg1					OSEKTime+Preemp+bus confg2				
Method	**Iter**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**	**# Binary**	**# Constraints**	**Time(s)**	**result**	**Opt**
(1)		50313	138944	3600	N/A	No	64907	174080	3600	50	No
(1)+(2)	1	50313	138955	3600	N/A	No	64907	174091	3600	N/A	No
(1)+(2)+(3)	1	35717	109173	3600	15	No	45983	133563	3600	45	No
	2	8361	30880	2.7	13	Yes	6439	24485	5.3	44	Yes
(1)+(2)+(3)+(4)	1	34909	106263	771.1	13	Yes	45175	130623	1952.7	44	Yes

when the optimum is found or there is no integer solution found in the time limit. Algorithm 1 is the corresponding script for implementation in AMPL.

Algorithm 1: Command Script to be implemented in AMPL

```
for each ECU $e_i$ do
    $k_i = \min(\sum A_{e_i,k}) = \text{Optimal}(F_{e_i})$ for $e_i$
end for
add constraints $\sum A_{e_i,k} \geq k_i, \forall e_i$ to $F$
repeat
    solve problem $F$ with time limit
    for each ECU $e_i$ do
        if $\sum A_{e_i,k} = k_i$ then
            fix the signal to frame packing for $e_i$
        end if
    end for
until optimal solution found or solver stops prematurely
    without an integer solution
```

For our case studies, the solver finds the optimal solutions within an hour for two of the configurations, even though the number of binary variables after the AMPL presolve phase remains the same [Method (1)+(2) in Table III].

For the system configuration with OSEK and bus configuration 2, the solver cannot find the optimal solution, but only a feasible solution using 46 slots. The new problem where the slot assignment is fixed for the ECUs that reached their lower bound has 2404 binary variables and can be solved in 1305.2 s. Similarly, in OSEKTime, when scheduling without preemption for the second type of bus configuration, the solver cannot find the optimal solution in an hour. In the second iteration there are 4344 binary variables and the problem can be solved in 9.7 s. In both cases, the optimal solution is found after the second iteration (44 used slots).

However, optimality cannot be guaranteed when using the iterative procedure. For example, in the OSEKTime case with preemption, the solver fails to find a feasible solution within the time limit of 1 hour.

2) Time Slicing: The size of the problem can also be reduced by restricting the execution window assigned to each job and signal (as in [24]). Starting from the system dataflow as in Fig. 3, we define a directed acyclic graph where nodes represent jobs or signals exchanged between jobs on different ECUs. Edges link signals to their sender and receiver jobs. In the resulting graph, each edge (o_i, o_j) represents a precedence constraint where o_i needs to finish execution or transmission before the activation or start of o_j.

We assign a weight v_i to each task and signal representing its estimated response time. For jobs on OSEKTime systems, v_i is the worst-case computation time. On OSEK systems, v_i is the worst-case response time (16), i.e., $v_i = R_i$. For signals, v_i is the sum of several terms: the length of the dynamic segment, the symbol and the network idle time windows, the average distance of two assigned slots from the same ECU, and the copy time required by the signal at the sender and the receiver nodes. By such an assignment, if the window can accommodate v_i, there is a high probability to be able to schedule all tasks and signals.

The start point $t_i^{\min}$ and end point $t_i^{\max}$ of the time window assigned to each task and signal are defined to distribute evenly the slack time before the deadline on each computation path. Consider the example of Fig. 6. There are three objects in this path $\tau_1 \rightarrow \sigma_2 \rightarrow \tau_3$, with v_i equal to 15, 5, 11, respectively. The deadline of this path is set to be 40. The starting point $t_i^{\min}$ for the time window associated to each object in the path is computed assuming the phase Φ_3 of the sink object equal to zero. In this case, the laxity (amount of slack time) of the path is $l = 40 - (\Phi_1 + 15 + 5 + 11) = 9 - \Phi_1$. If we evenly distribute

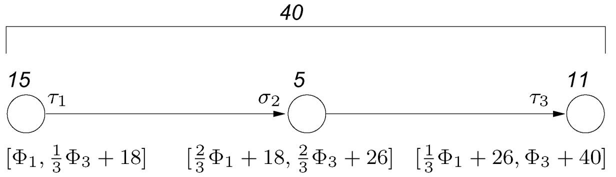

40
15
5
11
τ_1
σ_2
τ_3
$[\Phi_1, \frac{1}{3}\Phi_3 + 18]$
$[\frac{2}{3}\Phi_1 + 18, \frac{2}{3}\Phi_3 + 26]$
$[\frac{1}{3}\Phi_1 + 26, \Phi_3 + 40]$

Fig. 6. An example of time slice assignment.

the laxity to the objects in the path, the lower bounds of the time windows are

$$\begin{aligned} t_1^{\min} &= \Phi_1 \\ t_2^{\min} &= t_1^{\min} + 15 + \frac{1}{3}l = \frac{2}{3}\Phi_1 + 18 \\ t_3^{\min} &= t_2^{\min} + 5 + \frac{1}{3}l = \frac{1}{3}\Phi_1 + 26. \end{aligned} \tag{46}$$

To calculate the finish time $t_i^{\max}$ of each time window, we assume the phase Φ_1 of the source object is zero. Thus, the path laxity is $l' = 40 + \Phi_3 - (15 + 5 + 11) = 9 + \Phi_3$ and, starting from the sink object, we calculate $t_i^{\max}$ as follows:

$$\begin{aligned} t_3^{\max} &= \Phi_3 + 40 \\ t_2^{\max} &= t_3^{\max} - 11 - \frac{1}{3}l' = \frac{2}{3}\Phi_3 + 26 \\ t_1^{\max} &= t_2^{\max} - 5 - \frac{1}{3}l' = \frac{1}{3}\Phi_3 + 18. \end{aligned} \tag{47}$$

This method preserves the precedence constraints and evenly distributes the slack time to the tasks and signals in the path (it maximizes the minimum laxity of the time windows). The schedule obtained with this method is possibly not optimal because of the additional constraints, but is likely to be an extensible one, given the distribution of the slack time along each computation path.

Shown as Method (1)+(2)+(3) in Table III, by adding time slices to tasks and signals on top of step (2) we can further improve the solver efficiency, reducing the number of binary variables and its run time. For example, in the case study with OSEK and bus configuration 1, the number of binary variables is reduced by 20.8%, and the run time is shortened by 15.2%. More importantly, for the six case studies the optimal solution after this step remains the same, which demonstrates that imposing time windows to tasks and signals allows to reduce the search space without necessarily yielding on the quality of the solution.

3) Bounding Signal Start Times and Task Phases: Given the definition of the weights v_i and of the execution windows $[t_i^{\min}, t_i^{\max}]$ as in the previous section, the start time and finish time for each signal σ_i satisfy the following constraints:

$$\begin{aligned} s_i &\geq t_i^{\min} + \gamma_{l,i} \\ f_i &\leq t_i^{\max} - \gamma_{i,j}. \end{aligned} \tag{48}$$

For a task i

$$f_i \leq t_i^{\max}. \tag{49}$$

Also, if task i is an OSEKTime task

$$s_i \geq t_i^{\min} \tag{50}$$

or if it is an OSEK task

$$A_i \geq t_i^{\min} - J_i. \tag{51}$$

Conditions (48)–(51) and (28)–(32) can be used to find the bounds for task phases and signal start times ($t_i^{\min}$ and $t_i^{\max}$ are functions of the phases of the tasks at both ends of the path). Algorithm 2 gives the details of calculating the bounds for tasks and signals. First, the bounds of task phases and signal start times are initialized. Then, according to the constraints in (48)–(51) and (28)–(32), the lower bound is propagated from the sources to the sinks in the precedence graph (line 8–18), and the upper bound is propagated in the reverse order (line 19–29).

Algorithm 2: Find Bounds for Task Phases and Signal Start Times

```
1: for each task $\tau_i$ do
2:   $\Phi_i^{\min} = 0$, $\Phi_i^{\max} = d_i - v_i$
3: end for
4: for each signal $\sigma_i$ do
5:   $s_i^{\min} = 0$, $s_i^{\max} = t_{\max}$
6: end for
7: $V' = \mathrm{TopologicalOrdering}(V)$ of precedence graph
8: for $o_i$ in $V'$ from first element to last do
9:   for each outgoing edge $(o_i, o_j)$ do
10:      if $o_i$ and $o_j$ are tasks then
11:         $\Phi_j^{\min} = \max(\Phi_j^{\min}, \Phi_i^{\min} + v_i)$
12:      else if $o_i$ is a task and $o_j$ is a signal then
13:         $s_j^{\min} = \max(s_j^{\min}, \Phi_i^{\min} + v_i + \gamma_{i,j})$
14:      else if $o_i$ is a signal and $o_j$ is a task then
15:         $\Phi_j^{\min} = \max(\Phi_j^{\min}, s_i^{\min} - \gamma_{l,i} + v_i)$
16:      end if
17:   end for
18: end for
19: for $o_j$ in $V'$ from last element to first do
20:   for each incoming edge $(o_i, o_j)$ do
21:      if $o_j$ and $o_i$ are tasks then
22:         $\Phi_i^{\max} = \min(\Phi_i^{\max}, \Phi_j^{\max} - v_i)$
23:      else if $o_j$ is a task and $o_i$ is a signal then
24:         $s_i^{\max} = \min(s_i^{\max}, \Phi_j^{\max} - v_i + \gamma_{l,i})$
25:      else if $o_j$ is a signal and $o_i$ is a task then
26:         $\Phi_i^{\max} = \min(\Phi_i^{\max}, s_j^{\max} - v_i)$
27:      end if
28:   end for
29: end for
```

Once the bounds of signal start times are found, we can add these bounds as constraints to the problem, as follows:

$$\forall \tau_i, \Phi_i^{\min} \leq \Phi_i \leq \Phi_i^{\max} \quad (52)$$

$$s_i^{\min} \leq s_i \leq s_i^{\max}. \quad (53)$$

More importantly, we can predefine many signal to slot mapping variables $A_{i,j,k}$ as in (54) and (55). This is helpful for the solver to trim the solution space without losing optimality

$$s_{j,k}^s < s_i^{\min} \Rightarrow A_{i,j,k} = 0 \quad (54)$$

$$s_{j,k}^s > s_i^{\max} \Rightarrow A_{i,j,k} = 0. \quad (55)$$

This additional improvement, together with the other methods, allows to shorten the running time of the solver [Method (1)+(2)+(3)+(4) in Table III] by further reducing the number of binary variables. With these methods, all six configurations in the table can be solved in less than one hour. The optimal solution is found in all cases and it is the same for all of them.

C. Scalability Analysis

To study the scalability of the proposed formulation for the slot minimization objective, we generated a set of case studies by randomly changing the case study in Tables I and II.

The first scalability analysis is performed with respect to the number of signals in the design, by removing or adding signals to the case study in Tables I and II. Each new experiment is generated by adding or removing a set of eight signals to the case study in the previous section, with size and period distribution as in the Table II. Overall, we generate configurations with a number of signals going from 100 to 204. The set of tasks and ECUs is the same as in the original case study, and we choose the OSEK scheduling for tasks and the bus configuration 1. The results are given in Fig. 7. As shown in the top figure, in two of the cases the number of used slots returned by the MILP solver is one slot larger than the lower bound provided by (45). On the other hand, the runtime of the solver is dependent on the time limit and the number of iterations in Algorithm 1. In our experiments, we set a time limit for each iteration as 3600 s. For cases with a number of signals ≤156, the solver can find the optimal solution within the time limit, achieving the lower bound in (45) in the first iteration. For the cases with 196 or 204 signals, the solver returns a result significantly larger than the lower bound during the first iteration, and three more iterations are needed to finish the optimization process. In these two cases, the best solution found at the end of the optimization uses one more slot than the lower bound. Therefore, it is possible that the solver computes a suboptimal solution. To improve the situation, the time limit at each iteration could be increased in the hope that the solver finds a better quality solution in the early iterations at the price of longer running times.

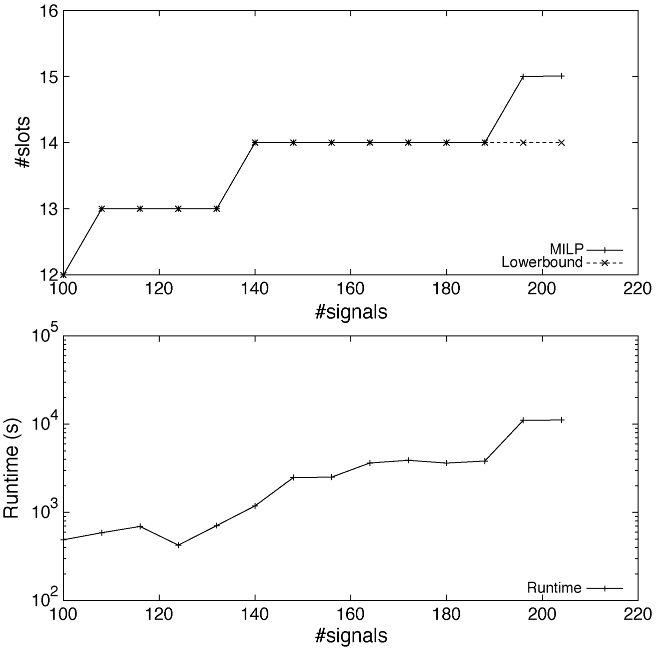


Fig. 7. Number of used slots and runtime of MILP solver versus number of signals.

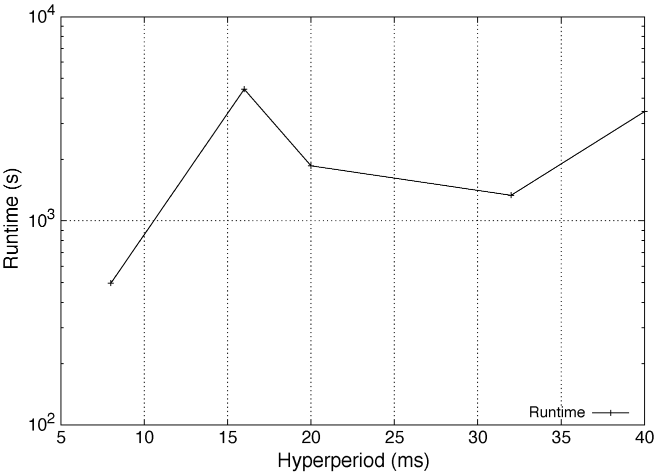


Fig. 8. Runtime of MILP solver versus hyperperiod.

We also study the impact of periods on the complexity of the problem. To do so, the task and signal periods are randomly selected from two sets of harmonic periods: $\{1, 2, 4, 8, 16, 32\}$ ms, and $\{2.5, 5, 10, 20, 40\}$ ms. Correspondingly, the hyperperiod of the system may take values from the set {8, 16, 20, 32, 40} ms. The set of tasks, signals and ECUs is the same as in the original case study, and OSEK is still the scheduling policy of choice for tasks. Also, configuration 1 (with $\text{communication cycle} = 1\ \text{ms}$) is selected for the FlexRay bus. In these experiments, the solver always achieves the lower bound in (45) at the end of the iterative processes. The runtime of the solver is plotted in Fig. 8, where the x axis is the hyperperiod of the system. From the figure, the runtime-complexity that is, the runtime of the solver, increases with the length of the hyperperiod, but in a moderate way. For example, the runtime is about 3434 s (less than one hour) when the hyperperiod increases to 40 ms.

The scalability analysis presented here only covers a small portion of the possible extensions. There are so many dimensions to the problem that it is not possible to define experimentally the region of applicability. The number of ECUs, the number of tasks and their periods and computation times, the number of signals and the topology of the functional communication and, of course, the size of signals, all come into play.

Exploring exhaustively all these dimensions or even a significant subset is unpractical, especially considering that for problems of industrial size, optimization times of several hours are expected.

Our limited analysis on two of the possible application dimensions (number of signals, task periods) provides useful information and hints to what can happen when our methodology is applied to (much) larger case studies. In Fig. 7, the runtime increases exponentially with the number of signals. A straightforward application of the proposed MILP technique to systems with thousands of signals could be not feasible or provide only a feasible solution, but not the optimum. In this case, it is still possible to use our approach incrementally, by dividing the application into task and message subsets, possibly using (time and functional) criticality as a classifier. In this case, the application subsets could be manageable and the problem could be solved iteratively starting from the most critical components and moving to less critical ones after the previous layers have been allocated to the task and message scheduling tables. This would require solving several consecutive MILP problems for each application subset or criticality class. Of course, this method cannot guarantee optimality and (in the worst case) not even finding a feasible solution if the problem admits one. However, such an incremental approach should provide a good tradeoff between runtime and optimality.

VII. Experimental Results: Maximizing the Minimum Laxity

Some of the reduction techniques presented in Section VI-B are specific to the selected cost function (minimizing the number of used slots), and are not applicable to other objectives. In this section, we discuss a different optimization metric, related to performance and robustness with respect to timing constraints: maximizing the minimum laxity (difference between end-to-end deadlines and latencies) among a set of paths selected by the designer, as in (44). In the following, we present a set of reduction techniques to make the optimization process capable of solving industrial-size problems in a reasonable amount of time.

A. Restricting the Search Space

1) Identification of Critical Paths: We first determine an upper bound on the maximum laxity of each of the selected paths. The laxity of a path cannot be larger than the difference between the deadline and the amount of time spent for computation and communication by the tasks and signals on the path. We use the term w_i for each task and signal representing the minimum guaranteed delay from its arrival to its finish. For jobs on OSEKTime systems, w_i is the worst-case execution time. On OSEK systems, w_i is the worst-case response time, i.e., $w_i = R_i$. For signals, w_i is the length of a static slot, plus the copy time required by the signal at the sender and the receiver nodes.

The maximum laxity or *slack* q_p for a path p can be calculated as

$$q_p = D_p - \sum_{i \in p} w_i. \tag{56}$$

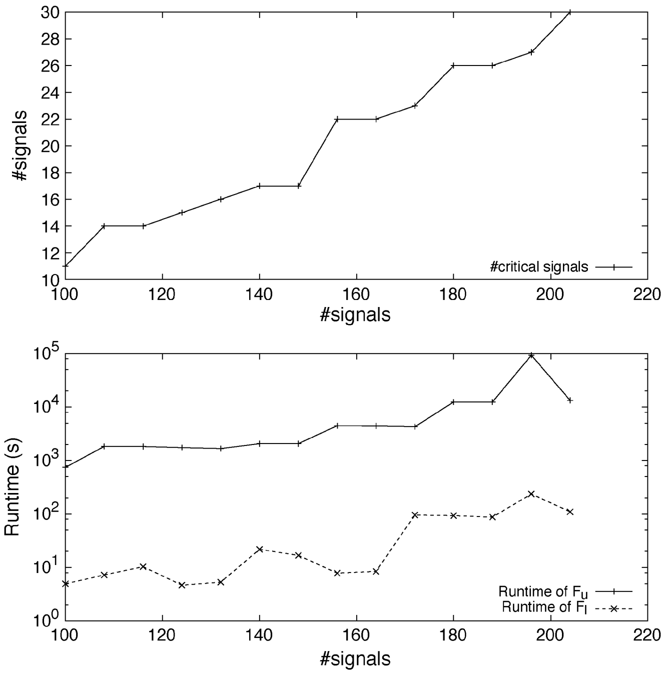


Fig. 9. Number of critical signals and runtime of MILP solver versus number of signals.

Among all selected paths, the ones with lower slack are likely to be also the ones with lower laxity, although it is not necessarily true that the path with minimum slack is the one with minimum laxity in the optimal solution. Thus, we define a set of *critical paths*, for which the slack is within a certain range (e.g., the average distance of two assigned slots from the same ECU) of the minimum slack.

At this point, we consider a relaxed version F_u of the original problem F, containing only the signals belonging to the *critical paths* identified in the previous step. The problem is usually much simpler than F. If the solver finds a solution for F_u, the corresponding optimal value is a (tight) **upper bound** on the optimal solution of the original problem F.

2) Prepacking of Signals: If two signals have the same period, sender, and receiver tasks in the selected paths, then the minimum laxity remains the same if we exchange the slot assignment of the two signals (assuming the slot size is large enough to allow so). We define such two signals as *bound* to each other. We propose a preprocessing stage to bound the prepackable signals together. To allow enough freedom of signal to slot assignment in F_l, a restriction can be enforced on the size of the bounded signals, e.g., smaller than one third of the static slot size.

The optimization process is the following. First, the problem F_u is solved and the optimal solution z_u is recorded. Then, a problem F_l is generated with the following additional constraints to F:

- the slot assignment of signals in the critical paths is fixed according to the solution of F_u;
- the signals which are not in the critical paths are bound together;
- the objective function is no greater than z_u.

TABLE IV
LIST OF PARAMETERS

Parameter	Implication	Domain
τ_i	the i-th task	
e_i	ECU resource on which task τ_i executes	
T_i	period of task τ_i	$\mathbb{N}$
J_i	release jitter of task τ_i	$\mathbb{R}^+$
C_i	execution time of task τ_i	$\mathbb{R}^+$
d_i	deadline for task τ_i	$\mathbb{R}^+$
P_i	priority level of an (OSEK) task τ_i	$\mathbb{R}^+$
R_i	worst-case response time of an (OSEK) task τ_i	$\mathbb{R}^+$
nc_i	number of jobs in one hyperperiod for task τ_i	$\mathbb{N}$
$hp(i)$	set of tasks with priority higher than τ_i	
$\sigma_{h,k}$	data signal between tasks τ_h and τ_k	
$b_{h,k}$	bit width of signal $\sigma_{h,k}$	$\mathbb{N}$
σ_i	data signal with global index i	
src(σ_i)	sender job for signal σ_i	
dst(σ_j)	receiver job for signal σ_j	
$t_{k,j}$	j-th job of the task τ_k	
t_i	job with the global index i	
r_i	worst-case response time of job t_i	$\mathbb{R}^+$
$r_{k,j}$	worst-case response time of job $t_{k,j}$	$\mathbb{R}^+$
H	length of application cycle (*lcm* of task periods)	$\mathbb{N}$
l_{comm}	length of the FlexRay communication cycle	$\mathbb{N}$
n_{slot}	no. of static slots in the communication cycle	$\mathbb{N}$
l_{slot}	length of a static slot in time	$\mathbb{N}$
b_{slot}	size of a slot in bits	$\mathbb{N}$
m_i	frame transmitted over a slot	
$s^s_{k,j}$	start time of j-th static slot in k-th comm cycle	$\mathbb{N}$
$f^s_{k,j}$	finish time of j-th static slot in k-th comm cycle	$\mathbb{N}$
$\mathbb{P}$	set of latency sensitive paths	
$P_{i,j}$	a path from job t_i to t_j	
src(p)	source job of path p	
snk(p)	sink job of path p	
D_p	end-to-end deadline of a path p	$\mathbb{R}^+$
M	a sufficiently large constant	$\mathbb{R}^+$
θ	set of jobs on the same ECU with no dependency	
$\gamma_{i,j}$	the worst-case copy time from o_i to o_j	$\mathbb{R}^+$
$t_{\max}$	the maximum length of the scheduling window; computed as $H + \max_i(T_i)$	$\mathbb{R}^+$
n_i	number of jobs of task τ_i within $t_{\max}$	$\mathbb{N}$
n_a	number of communication cycle within $t_{\max}$	$\mathbb{N}$
n_l	number of slots in n_a-th cycle within $t_{\max}$	$\mathbb{N}$

TABLE V
LIST OF VARIABLES

Variable	Implication	Domain
Φ_i	initial phase of task τ_i	$\mathbb{R}^+$
a_i	arrival time of job t_i	$\mathbb{R}^+$
$a_{k,j}$	arrival time of j-th job of the task τ_k	$\mathbb{R}^+$
A_i	release time of job t_i	$\mathbb{R}^+$
J_i	jitter of task τ_i	$\mathbb{R}^+$
s_i	start time of job t_i or signal σ_i	$\mathbb{R}^+$
f_i	finish time of job t_i or signal σ_i	$\mathbb{R}^+$
$y_{i,j}$	the order of start times between jobs t_i and t_j; true if $s_j \geq s_i$	$\mathbb{B}$
$p_{i,j}$	preemption between jobs t_i and t_j; true if job t_i is preempted by job t_j	$\mathbb{B}$
$A_{i,j,k}$	mapping of signals to slots; true if signal σ_i is mapped to k-th slot of j-th cycle	$\mathbb{B}$
$A_{e_i,j}$	the ownership of a slot by an ECU; true if slot j is owned by ECU e_i	$\mathbb{B}$

F_l is solved with an optimal solution z_l. The solution to F_l (when computed by the solver) provides a **lower bound** on the actual optimal solution of the original problem F. If $z_u = z_l$, then the optimal solution is found; otherwise, the original problem F is solved with additional constraints that the objective function is within the range of $[z_l, z_u]$.

B. Scalability Analysis

We applied the proposed optimization process to the same set of system configurations used for the scalability analysis in Section VI-C. We used OSEK scheduling for tasks and the bus configuration 1, and assumed that *all paths in the system with deadlines* are sensitive to latency and are considered in the metric function. For the original system in Tables I and II, there are a total of 357 paths, while the number of paths is 877 for the system with 196 signals. On the top diagram of Fig. 9, the number of signals in the critical paths is plotted. Despite the large number of paths, this number is much smaller than the total number of signals.

For all the case studies in the scalability analysis, the two bounds z_u and z_l returned by solving the problems F_u and F_l are the same, thus the optimization process stops after the solution of F_l and the optimal objective function is $z_u = z_l$. The runtimes for solving the simpler problems F_u and F_l are shown in the bottom diagram of Fig. 9. The optimal solution to problem F_u is typically found in a very short time (always within an hour), but it can take a very long time (more than one day in the case with 196 signals) to prove its optimality. Also, the problem F_l can be solved very efficiently. This is because the assignment of the slots to critical signals is fixed, and the packing of the other signals is simplified by leveraging the observation that the objective function is not sensitive to their scheduling.

VIII. CONCLUSION

This paper presents an optimization framework based on an MILP formulation to schedule transactions consisting of tasks and signals on a FlexRay-based system. The objective of the proposed MILP method is to maximize the number of free communication slots (therefore improving extensibility) or to maximize minimum laxity among paths (therefore improving timing performance). We provide solutions for the synchronized task-to-signal information passing under different task scheduling policies based on existing industry standards. The optimal solutions can be found within reasonable time for our industrial-size case studies, which demonstrate the efficiency of our problem formulation on real systems and the effective interaction with the MILP solver.

APPENDIX
LIST OF SYMBOLS, PARAMETERS, AND VARIABLES

The list of parameters and variables in the MILP formulation are summarized in Tables IV and V respectively. The symbols, with their implications and, if appropriate, domains are listed in the tables. The possible domains of parameters and variables include: booleans (denoted as $\mathbb{B}$), non-negative real numbers (denoted as $\mathbb{R}^+$), and naturals (denoted as $\mathbb{N}$).

REFERENCES

[1] *"The Mathworks Simulink User Manual."* [Online]. Available: http://www.mathworks.com

[2] *Iso 11898-1. Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*, ISO Standard-11898., International Standards Organization (ISO), Nov. 1993.

[3] *Road Vehicles-Controller Area Network (CAN) Part 4: Time-Triggered Communication*, ISO Standard-11898-4, International Standards Organization (ISO), 2004.

[4] Osek/vdx Operating System Specification ver. 2.2.3, Feb. 2005. [Online]. Available: http://www.osek-vdx.org

[5] K. J. Astrom and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1997.

[6] J. Berwanger, M. Peller, and R. Griessbach, "Byteflight—A new high-performance data bus system for safety-related applications." 2004. [Online]. Available: http://www.byteflight.com

[7] J. Broy and K. D. Muller-Glaser, "The impact of time-triggered communication in automotive embedded systems," in *Proc. SIES Conf.*, 2007, pp. 353–356.

[8] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From simulink to scade/ lustre to TTA: A layered approach for distributed embedded applications," *SigPlan Not.*, vol. 38, no. 7, pp. 153–162, 2003.

[9] G. Cena and A. Valenzano, "Performance analysis of byteflight networks," in *Proc. IEEE Int. Workshop on Factory Commun. Syst.*, 2004, pp. 157–166.

[10] The Autosar Consortium. Autosar Flexray Interface Driver Specification ver. 4.0. [Online]. Available: www.autosar.org, 2010

[11] IBM CPLEX. [Online]. Available: http://www.ibm.com/software/integration/optimization/cplex-optimizer/ ver. 11.0

[12] S. Ding, N. Murakami, H. Tomiyama, and H. Takada, "A GA-based scheduling method for flexray systems," in *Proc. 5th EMSOFT Conf.*, 2005, pp. 110–113.

[13] P. Eles, Z. Peng, P. Pop, and A. Doboli, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 5, pp. 472–491, 2000.

[14] J. Ferreira, P. Pedreiras, L. Almeida, and J. A. Fonseca, "The FTT-CAN protocol for flexibility in safety-critical systems," *IEEE Micro*, vol. 22, no. 4, pp. 46–55, 2002.

[15] FlexRay. Flexray Communications System Protocol Specification, ver. 2.1, revision a. [Online]. Available: http://www.flexray.com

[16] M. Grenier, L. Havet, and N. Navet, "Configuring the communication on flexray: The case of the static segment," in *Proc. 4th Eur. Congr. Embedded Real Time Software: ERTS'08*, Toulouse, France, Jan. 29–Feb. 1 2008.

[17] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh, "Performance analysis of flexray-based ECU networks," in *Proc. 44th DAC Conf.*, 2007, pp. 284–289.

[18] A. Hamann and R. Ernst, "TDMA time slot and turn optimization with evolutionary search techniques," in *Proc. DATE Conf.*, 2005, pp. 312–317.

[19] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, and R. Ernst, Symta/s—Symbolic Timing Analysis for Systems. [Online]. Available: http://www.symta.org 2004

[20] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. IEEE*, pp. 112–126, 2003.

[21] K. Lakshmanan, G. Bhatia Bhatia, and R. Rajkumar, "Integrated end-to-end timing analysis of networked autosar-compliant systems," in *Proc. 13th Design, Autom. Test in Europe Conf. (DATE)*, 2010, pp. 331–334.

[22] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," in *Proc. 7th IEEE/ACM Int. Conf. Hardware/Software Codesign and System Synthesis; CODES+ISSS'09:* , 2009, pp. 363–372.

[23] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Manage. Sci.*, vol. 44, no. 3, pp. 388–399, 1998.

[24] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proc. IEEE Real-Time Systems Symp.: RTSS'94:* , 1994, pp. 216–227.

[25] M. Di Natale and H. Zeng, "Time determinism and semantics preservation in the implementation of distributed functions over flexray," *Society of Automotive Eng. World Congr.*, 2010.

[26] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.

[27] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time-triggered embedded systems," *Real-Time Systems*, vol. 24, pp. 297–325, 2004.

[28] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the flexray communication protocol," in *Proc. ECRTS'06*, 2006, pp. 203–399.

[29] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the flexray communication protocol," *Real-Time Syst.*, vol. 39, no. 1–3, pp. 205–235, 2008.

[30] J. M. Rushby, "Bus architectures for safety-critical embedded systems," in *Proc. 1st Int. Workshop on Embedded Software: EMSOFT '01*, 2001, pp. 306–323.

[31] K. Schmidt and E. G. Schmidt, "Message scheduling for the flexray protocol: The static segment," *IEEE Trans. Veh. Technol.*, vol. 58, no. 5, pp. 2170–2179, 2008.

[32] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, and M. Di Natale, "Implementing synchronous models on loosely time-triggered architectures," *IEEE Trans. Computers*, vol. 57, no. 10, pp. 1300–1314, Oct. 2008.

[33] H. Zeng, W. Zheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli, "Scheduling the flexray bus using optimization techniques," in *Proc. 46th Annu. Design Autom. Conf.: DAC '09*, 2009, pp. 874–877.

Haibo Zeng received the B.E. and M.E. degrees in electrical engineering from Tsinghua University, Beijing, China, and the Ph.D. degree in electrical engineering and computer sciences from University of California at Berkeley.

He is currently a Senior Researcher with General Motors R&D. His research interests cover design methodology, analysis and optimization for real-time systems and embedded systems.

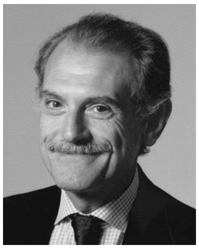
Marco Di Natale (M'03) received the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, Italy, in 1991.

He is an Associate Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, in which he held a position as Director of the Real-Time Systems (ReTiS) Lab from 2003 to 2006. He has been a Visiting Researcher at the University of California, Berkeley, in 2006 and 2008/09. He has been selected in 2006 by the Italian Ministry of Research as the national representative in the mirror group of the ARTEMIS European Union Technology platform. He has been a researcher in the area of real-time systems and embedded systems for more than 15 years, being author or coauthor of more than 100 scientific papers.

Dr. Di Natale received three Best Paper Awards and the Archie T. Colwell Award. He has served as Program Committee member and has been organizer of tutorials and special sessions for the main conferences in the area, including the Real-Time Systems Symposium, the IEEE/ACM Design Automation Conference (DAC), the Design Automation and Test in Europe (DATE), and the Real-Time Application Symposium. He also served as Track Chair for the RTAS Conference, the Automotive Track of the 2010 DATE Conference. He has been an Associate Editor for the IEEE TRANSACTIONS ON CAD and the IEEE EMBEDDED SYSTEMS LETTERS and is currently on the Editorial Board of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.

Arkadeb Ghosal received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kharagpur, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley.

His research interests cover design, analysis, comparison and optimization for real-time systems and embedded systems. He is currently a Staff Software Engineer at National Instruments Engineering Berkeley, CA.



Alberto Sangiovanni-Vincentelli (F'81) received the electrical engineering and computer science degree ("Dottore in Ingegneria") (*summa cum laude*) from the Politecnico di Milano, Milano, Italy, in 1971.

He holds the Edgar L. and Harold H. Buttner Chair of Electrical Engineering and Computer Sciences at the University of California at Berkeley. In 1987, he was Visiting Professor at MIT. He was a co-founder of Cadence and Synopsys, the two leading companies in the area of electronic design automation. He is a member of the Board of Directors of Cadence and the Chair of its Technology Committee, UPEK, Sonics, and Accent. He was a member of the HP Strategic Technology Advisory Board, and is a member of the Science and Technology Advisory Board of General Motors. He is a member of the High-Level Group, of the Steering Committee, of the Governing Board and of the Public Authorities Board of the EU Artemis Joint Technology Undertaking. He is member of the Scientific Council of the Italian National Science Foundation (CNR) and a member of the Executive Committee of Italian Institute of Technology. He is an author of over 800 papers, 15 books and 3 patents.

Dr. Sangiovanni-Vincentelli is a Member of the National Academy of Engineering. He received the Kaufman Award from the Electronic Design Automation Council for "Pioneering Contributions to EDA" and the IEEE/RSE Wolfson James Clerk Maxwell Medal for groundbreaking contributions that have had an exceptional impact on the development of electronics and electrical engineering or related fields.