

Designing FlexRay-based Automotive Architectures: A Holistic OEM Approach

Paul Milbredt[†], Michael Glaß^{*}, Martin Lukasiewicz^{*}, Andreas Steininger[‡], and Jürgen Teich^{*}

[†] AUDI AG, Germany, paul.milbredt@audi.de

^{*} University of Erlangen-Nuremberg, Germany, {glass,teich}@cs.fau.de

^{*} TUM CREATE, Singapore, martin.lukasiewicz@tum-create.edu.sg

[‡] Vienna University of Technology, Austria, steininger@ecs.tuwien.ac.at

Abstract—FlexRay is likely to become the de-facto standard for upcoming in-vehicle communication. Efficient scheduling of the static and dynamic segment of the communication cycle in combination with the determination of more than 60 parameters that are part of the FlexRay protocol is a challenging task.

This paper provides a formal analysis for interdependencies between the parameters as well as a scheduling approach for the static and dynamic segment. Experimental results give evidence of a significant interdependency between the subtasks such that a holistic scheduling approach becomes mandatory to provide high-quality FlexRay schedules. As a solution, this work introduces a complete functional FlexRay scheduling approach that takes parameter selection, allocation of messages to the static and dynamic segment, and concurrent scheduling into account. A real-world case study from the automotive domain gives evidence of efficiency and applicability of the proposed approach.

I. INTRODUCTION

Design and system integration are the most critical tasks of an automotive original equipment manufacturer (OEM). In modern automobiles, more than 80 *Electronic Control Units* (ECUs) are connected via different buses. Upcoming applications like advanced driver assistance systems require a higher bandwidth than provided by well-established field buses like CAN. As a remedy, an industrial consortium [3] developed the *FlexRay* bus. It offers a bandwidth of up to 10 Mbps and features two bus access schemes in the *static segment* and *dynamic segment*, respectively. In the static segment, messages are scheduled strictly time-triggered into *slots* of equal size such that a deterministic transmission is guaranteed. In the dynamic segment, the transmission is event-triggered into slots of variable size, resulting in a high utilization. The configuration of the FlexRay bus is a challenging task. More than 60 parameters that have to be chosen carefully and a feasible schedule has to be determined. These design decisions have a substantial influence on the utilization of the bus and performance of the system.

Designing FlexRay-based automotive systems is a complex problem. This complexity arises from the interdependency of the three required design steps. The (a) scheduling of the static segment, the (b) scheduling of the dynamic segment, and the (c) parameter determination. Previous work studied these steps separately. However, an OEM is required to resolve this interdependency to obtain an appropriate system implementation. In this work, a holistic and functional approach for the design of FlexRay-based automotive systems is proposed. The presented design approach integrates all design steps that are carried out by an OEM in a well-orchestrated fashion. A globally optimized FlexRay-based system is obtained where the determined schedules and parameters optimize the bus utilization. Additionally, the approach considers the *extensibility* of the schedules in order to consider future incremental extensions of the system. This is achieved by a proposed layer of abstraction for the static segment by assigning slots to virtual ECUs. For the dynamic segment, a special scheduling is proposed that takes different operation modes of an automobile into account and enables a guaranteed transmission of all data.

II. THE FLEXRAY PARAMETRIZATION PROBLEM

This section introduces the fundamentals of the FlexRay bus and the strong interdependency between the schedule and parameter set. As FlexRay is a time-triggered system for automotive applications, the main scheduling optimization goal is the determination of an efficient schedule that consumes as little communication time as possible. At the same time, an easily extensible schedule is desirable due to the long product life cycle of an automobile.

A. FlexRay Fundamentals

The FlexRay bus has a rather complex structure, see [4]: The global time is divided into 64 cycles of equal length. In practice, the typically chosen cycle length is 5 ms, resulting from the cycle times of existing CAN applications. Also, the typical length of a Protocol Data Unit (PDU) is eight byte to simplify a heterogeneous employment of FlexRay and CAN buses in the same system.

Each FlexRay cycle consists of the *static segment*, the *dynamic segment*, the *symbol window*, and the *network idle time*.

- The static segment consists of equally sized static slots that contain the same amount of payload. Each slot is assigned to exactly one or no ECU in all cycles.
- The optional dynamic segment consists of *minislots*. In contrast to the static segment, slots may be assigned to different ECUs in different cycles. Additionally, the frames within these slots may contain a varying payload of 0 – 254 bytes in two-byte steps.
- The optional symbol window may be used to transmit a *media test symbol*. This symbol is employed for testing the optional bus guardian. Currently, the symbol window is not used in any implementation of FlexRay clusters.
- The *network idle time* (NIT) enables the synchronization of the local clocks of the ECUs. This is mandatory for the correct behavior of the system.

1) *Timing hierarchy*: The basic time unit in FlexRay is the *microtick*. It is derived directly from the quartz of the node. All known FlexRay implementations use a microtick length of 25 ns while the specification would also allow 12.5 or 50 ns. To allow time-triggered communication, a globally synchronized time unit is necessary: the *macrotick*. In each node, the macrotick is built up of an integer number of microticks by a slightly modification of the Bresenham algorithm [1], the parameters used in the implementations are the number of microticks and macroticks per cycle. The macrotick length τ^{MT} is used in the specification, but is only a helper variable. The number of microticks that must pass before the macrotick counter is increased is influenced by the clock synchronization, which performs offset and rate correction, and the selection of the macrotick length. The macrotick length is required to be in the range between 1 and 6 μs and the cycle length must be an integer multiple of the macrotick. For a 5 ms cycle and a 25 ns microtick, the cycle length is 200,000 microticks. For a macrotick length of 1 μs , the cycle length is 5,000 macroticks, and for a length of 6 μs , it is $\left\lceil \frac{5,000}{6} \right\rceil = 834$ macroticks. This leads to 5,000 – 834 + 1 = 4,167 possible macrotick lengths for a 5 ms cycle.

All other timing related parameters are derived from the macrotick as integer multiples. These parameters are the *static slot size*, the *minislot size*, the *symbol window*, the *network idle time*, and the *action point offset*.

B. Influences of the Parameter Set on Bus Utilization

In the following, effects that create significant interdependencies between the scheduling and the parameter determination are selected and discussed. The key parameter in FlexRay is the macrotick.

1) *The Static Segment*: As a single static slot may only be assigned to one node, the whole duration of a static slot utilizes the bus even if only one bit must be transmitted. It is the task of the scheduling, to utilize the slots. But it is the task of the parametrization to minimize the duration of one slot. One static slot consists of three parts: (1) idle time at the beginning until the *action point offset*, (2) the transmission of the frame, and (3) idle time at the end. To avoid collisions, every transmission must be within the borders of the slot. To compensate clock drifts and the minimal physical propagation delay $\tau_{\text{min}}^{\text{prop}}$, the action point offset is calculated for worst case circumstances.

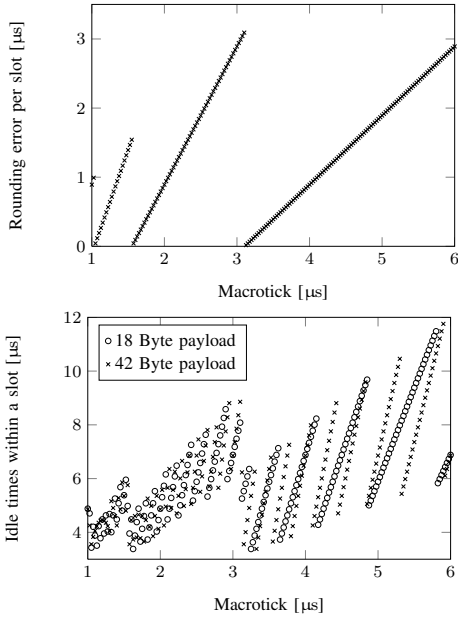


Fig. 1. Effect of rounding error for: (TOP) each action point offset for a given topology with a theoretical action point offset of 3.107791238 μs ; (BOTTOM) each static slot for a given topology for two given payload lengths.

FlexRay 2.1 specifies a maximum clock deviation d^c of 1,500 ppm in the network. As a result, the real minimal absolute length of a macrotick τ_{\min}^{MT} is given as follows:

$$\tau_{\min}^{\text{MT}} = \frac{\tau^{\text{MT}}}{1 + d^c} \quad (1)$$

The constraint 12 of the FlexRay specification [4] for the action point offset τ^{apo} must hold with respect to the shortest possible macrotick τ_{\min}^{MT} and the precision τ^{prec} :

$$\tau^{\text{apo}} \geq \left\lceil \frac{\tau^{\text{prec}} - \tau_{\min}^{\text{prop}}}{\tau_{\min}^{\text{MT}}} \right\rceil \quad (2)$$

The action point offset should be chosen as the smallest integer multiple of the macrotick fulfilling Equation 2. The resulting rounding error for a given example topology (a known precision and propagation delay) is depicted in Figure 1 top.

Besides the action point offset also the static slot size must be an integer multiple of the macrotick. The idle time at the end of the frame must be large enough to compensate imprecisions (like the action point offset) and additionally let all other nodes determine that the bus is idle (τ^{idle}). The length of the frame is obviously influenced by its payload length. The overhead is fixed and only the transmission start sequence is configurable between 3 and 15 bits. It is dependent on the topology and constrained by the specification. Usually, the smallest possible value is taken and then the frame length for a given payload length is constant.

Hence, for the whole static slot duration τ^{slot} , constraint 21 of the specification prevents boundary violations under worst case (with the maximum propagation delay $\tau_{\max}^{\text{prop}}$, but fault free) circumstances:

$$\tau^{\text{slot}} \geq \tau^{\text{apo}} + \left\lceil \frac{\tau^F + \tau_{\max}^{\text{prop}} + \tau^{\text{prec}} + \tau_{\max}^{\text{idle}}}{\tau_{\min}^{\text{MT}}} \right\rceil \quad (3)$$

with a frame length τ^F (including idle detection) of a frame consisting of $l^F(b)$ bits with a slow quartz, which generates bits of length τ_{\max}^{bit} :

$$\tau^F = \left(l^F(b) + 10 \right) \tau_{\max}^{\text{bit}} \quad (4)$$

For our given topology, the first rounding error occurs for the action point offset and the second for the whole slot for a given payload length. The resulting total error (idle time) is depicted in Figure 1 bottom for two of the 128 different possible payload lengths.

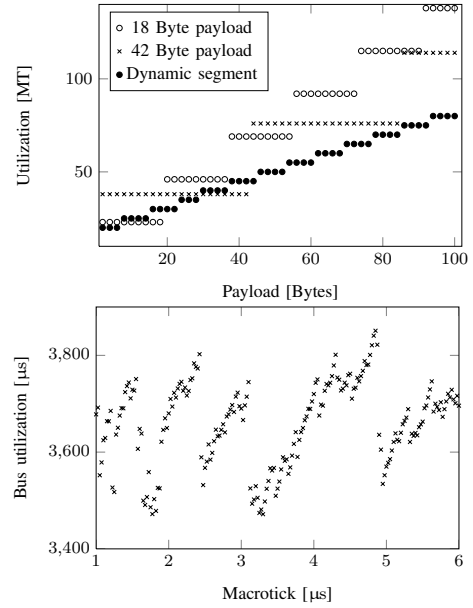


Fig. 2. Bus utilization of: (TOP) one ECU depending on its needed payload per cycle; (BOTTOM) depending on the chosen macrotick in a 5 ms cycle.

2) *The Dynamic Segment*: For the minislot action point offset, constraint 14 of the FlexRay specification holds. It is identical to constraint 12 given in Equation (2). The same applies to the length of the minislot, which is comparable to Equation (3) without the frame length.

If a node has data to transmit, the corresponding minislot is extended to a full slot with a length of an integer multiple of one *minislot*. It can be calculated with the helper variable `aMinislotPerDynamicFrame` of the FlexRay specification.

Additionally, different payload lengths are allowed. For determining the rounding error, the schedule must be known before. Then, the resulting error for the dynamic segment can be calculated for a given macrotick length.

An example for one node is given in Figure 2 top. Assume a node which needs to transmit 20 bytes of payload per cycle. In the case of scheduling its data in the static segment, two slots would have to be used, if the static payload was 18 bytes. If the payload was 42 bytes, a single slot, which is larger than the 18 byte slot, would be used. A dynamic frame could contain exactly 20 bytes of payload, but it depends on the chosen macrotick and derived parameters how many minislots this frame needs for transmission.

3) *Summarized Parametrization Effect*: For the whole effect of the parametrization, due to the large influence of the dynamic segment, the schedule must be known. Only then, the effective rounding errors of all slots can be calculated and minimized by selecting a good value for the macrotick.

The other way around, for a given parameter set, the optimal schedule might look differently. With our given topology, the selection of the optimal macrotick is counterintuitive. With the methodology presented during the remainder of this paper, a set of messages to be scheduled with no additional constraints on the macrotick is depicted in Figure 2 bottom.

III. RELATED WORK

Several approaches are proposed that focus on solely parts of the FlexRay scheduling problem: [10], [6], and [2] on the static segment, [9] on the dynamic segment scheduling. Other approaches, e.g., [12], investigate the combination of application task and bus scheduling. Timing analysis for the static segment is also presented in [8]. However, the approaches ignore the parametrization problem.

The prioritization of the frames of the dynamic segment is investigated in [11] where the assignment of PDUs within a frame is done in advance. Alternatively, each PDU equals one frame which is not optimal in terms of bus utilization. Since our assumption is that the response time of all frames is always lower than the cycle length, we extend the dynamic segment until all frames fit inside it. However, if no schedule can be found or our assumption does not hold, a combination of the approach in [11] and our approach becomes feasible.

The optimization of the parameter set has been investigated only in [7] so far. However, this work suffers from two major drawbacks: (1)

the macrotick is fixed at a length of 1 μ s. However, the macrotick is the key optimization parameter. (2) an optimization of the cycle length is performed, which in practical automotive applications is fixed at 5 ms due to the fixed minimal cycle lengths of automotive messages.

IV. FLEXRAY SCHEDULING

A. Bin Packing-based Static Segment Scheduling

The authors of [6] proposed an optimization approach, which considers cycle-multiplexing restricted to the static segment. As all static slots contain the same amount of payload, they transformed the problem to a special bin packing problem. Every slot is assumed to be a bin in which the PDUs are placed. The height of such an element corresponds to the period (repetition rate) of the message. In a 5 ms cycle, a PDU which must be scheduled every 5 ms is of full height, a 10 ms is of half height and so on. The width of an element is the payload, i.e., a 42 byte bin (slot) may contain five 8 byte 5 ms elements (PDUs) and their corresponding five update bits, leaving 11 bits unused.

Let the number of occupied slots be u^{Static} . The goal of static scheduling as proposed in [6] is to place the PDUs in such a way into slots that the number of used slots is minimized:

$$\text{minimize } u^{\text{Static}} \quad (5)$$

In this work, the methodology of [6] is extended to satisfy our considerations upon the parameter set. If only the static segment is taken into account, we would propose the following scheduling algorithm: (1) Perform bin-packing according to [6] for all possible payload lengths, (2) calculate the optimal macrotick for each possible payload. This is independent from the schedule, cf. Section II, and (3) select the schedule with lowest channel utilization. Within the context of this work, a modification of the aforementioned optimization criteria becomes necessary. Even a pure static segment solution still requires at least the network idle time for clock synchronization. The proposed static optimization criteria is to minimize the utilized time on the bus

$$\text{minimize } \left(u^{\text{Static}} \tau_{\text{Slot}} + \tau_{\text{NIT}} \right) \tau_{\text{MT}} \quad (6)$$

B. Bin Packing-based Dynamic Segment Scheduling

The dynamic segment offers three main advantages over the static segment: (1) A node with only small communication demands does not need to use a full static slot but can use a small dynamic slot instead (see Figure 2). (2) A node with large communication demands can use a large slot instead of multiple static slots. This saves overhead and, thus, bus utilization (see Figure 2). (3) A node which has no communication demand when its slot arrives, does not need to transmit and then uses only an idle minislot.

In particular, the last advantage can be used for functional extensions of the whole schedule. In different operation modes of the automobile, different *layouts* of the communication exist. Let one mode be the normal operation during driving. Here, e.g., large messages containing data from environmental sensors are transmitted. Another mode is the reprogramming of ECUs in the workshop. Here, no data from environmental sensors is transmitted, but only the binary of the new application is transmitted. As a consequence, the scheduler for the dynamic segment is supposed to determine a schedule such that each operation mode fits into the dynamic segment with guaranteed message transmission. As the communication is known at design time, it is possible to determine the minimum length of the dynamic segment for all modes. The length of the dynamic segment must be equal or larger than the longest layout of the communication. Note that the longest layout communication takes cycle-multiplexing into account, i.e., the length of the dynamic segment is not simply given by the length of all possible messages. Then, the transmission of messages in the dynamic segment is guaranteed, which makes the use of this segment feasible in the automotive domain.

For the usage of this segment, we have to consider two main disadvantages: (1) As it is not guaranteed which nodes, before a specific slot, do transmit, this leads to jitter of the frame due to shifts of slots. However, the earliest and latest possible transmission time can be given if the dynamic segment is chosen long enough as we proposed. (2) In FlexRay 2.1, noise during the dynamic segment leads to a corrupted data for the remaining cycle. This fault in the specification was fixed in FlexRay 3.0.

As the dynamic segment may contain frames of different length and also use cycle multiplexing for the slots, the bin packing transformation as proposed for the static segment is employed, adding two adoptions: (1) The dynamic segment is treated as a single bin with height 64 (i.e. cycles) and infinite length and (2) all elements (PDUs) can be placed inside at any position. After the solution of the bin packing problem has

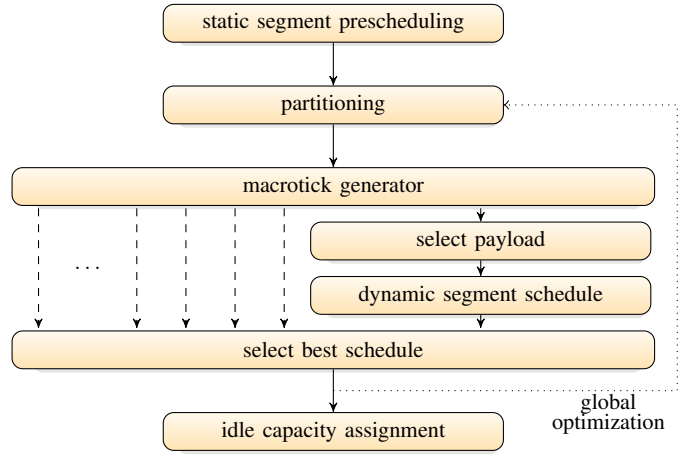


Fig. 3. Proposed overall design flow.

been calculated, the PDUs can be reordered by their sending nodes and their priority. Afterwards, every sender will have its elements in a single dynamic slot. In other words, the rectangle bin is cut, but the resulting slots must not be a rectangle, because a slot may have different payloads each cycle.

Compared to the static segment scheduling, the optimization criteria is slightly different. Only for the whole segment, the occupied area must be of a minimum size and for all cycles the length should not differ much. It is more common, that a highly periodic message will be scheduled in future versions, so the common length of all cycles ensures that this new bin of full height will fit into the schedule.

The dynamic segment is scheduled for every operation mode, as stated before. All operation modes build the set M . If in one mode m_1 slots are transmitted, which have a higher priority than some slots of mode m_2 , their minislots pass, which lengthens m_2 by these minislots. If they have lower priority than all slots in mode m_2 , their minislots do not appear, as they may be lost out. The utilized minislots $u^{\text{dyn}}(m, c)$ depend on the mode m , and on the other hand on the cycle $c \in \{0, \dots, 63\}$, because of the possibility to use different frame lengths and senders for each cycle.

The dynamic segment must be scheduled such that all modes fit into the segment. Its minimum length $\tau_{\text{min}}^{\text{dyn}}$ given in minislots is

$$\tau_{\text{min}}^{\text{dyn}} = \max_{\forall m \in M, c \in \{0, \dots, 63\}} u^{\text{dyn}}(m, c) \quad (7)$$

and the average utilization $u_{\text{avg}}^{\text{dyn}}$ is

$$u_{\text{avg}}^{\text{dyn}} = \frac{\sum_{c=0}^{63} \left(\max_{\forall m \in M} u^{\text{dyn}}(m, c) \right)}{64} \quad (8)$$

The length τ^{SW} of the symbol window is typically 0, as it is not used in any known implementation of FlexRay clusters. Combining all four segments, leads to holistic optimization criteria of minimizing

$$u = \frac{\tau_{\text{MT}} \left(u^{\text{Static}} \tau_{\text{Slot}} + u_{\text{avg}}^{\text{dyn}} \tau_{\text{Minislot}} + \tau^{\text{SW}} + \tau_{\text{NIT}} \right)}{\tau_{\text{Cycle}}} \quad (9)$$

C. Holistic Design Flow

The proposed overall design flow is outlined in Fig. 3. In a first step, several static schedules with reasonable slot sizes are generated. Afterwards, the PDUs are assigned to either the static or dynamic segment. Given the scheduling for different macroticks is independent of each other, the next steps investigate all possible macrotick sizes in parallel. In the same step, the payload size and dynamic schedule is determined as well. From the set of investigated schedules regarding slot size and macrotick, the schedule with lowest bus load is selected. At this point, a global optimization may be incorporated that aims at finding an optimal schedule by varying the partition of PDUs to static and dynamic segment. Since an exhaustive search for the optimal schedule is likely to be impracticable, a meta-heuristic search approach such as simulated annealing [5] is employed to control the optimization process.

V. EXPERIMENTAL RESULTS

We took a real world communication example from the chassis and powertrain domain. The data should be close to reality with only a short amount of uncertainty. Our data consists of 17 ECUs, sending out 264 PDUs in total. In addition to these application level PDUs, a number of network protocol related PDUs are distributed over the network, e.g., network management and transport protocols for diagnosis of the car.

The distribution of consumed bandwidth per ECU is as follows: One ECU is below 1%, seven ECUs consume 1%, two ECUs consume 3%, two ECUs consume 4%, two ECUs consume 5%, and the remaining four ECUs consume 9, 13, 18, and 30% bandwidth. The lowest amount of used bandwidth per ECU is 1,700 Bit/s, the highest amount is 325,900 Bit/s.

Only 3 ECUs, which come from a driver assistance application, make more than 61% of the used bandwidth whereas 9 ECUs consume less than 9% in total.

The more static slots a ECU consumes, the more overhead will be needed. On the other hand, if an ECU uses only one slot partly, the remaining space within the slot may only be consumed by this ECU and, according to our optimization criteria in Equation 9, it will produce non-optimal result. As a rule of thumb, ECUs with a huge amount of payload should be scheduled into the dynamic segment due to the reduction of overhead and ECUs with a small amount of payload, too, due to well utilized small slots.

Due to the fact that messages in the dynamic segment will jitter, e.g., because data has been added at the beginning of the dynamic segment in later versions of the schedule, but the ECU software is still “old”, or messages are not sent out and therefore consume only one minislot. So it must be up to the system designer to decide whether PDUs may be scheduled in the dynamic segment or not. In our case, the two ECUs with the highest amount of sent data and the ECU with the least amount of data to be sent may be completely scheduled dynamically. For all other ECUs, their application messages must be scheduled statically. All network protocols may be scheduled in the dynamic segment, and we already planned enough payload for later extensions of ECUs for them.

The application PDUs sum up to 1,074,100 Bits/s. Additionally, the protocol related messages (including messages used by developers) sum up to 2.5 MBit/s, which is not surprising, as these data is used for reprogramming the ECUs, testing purposes, maintenance, etc.

151 of the application PDUs are, due to the ability to route them to a CAN bus, of eight byte length, only two PDUs are of ten bytes length. All other PDUs are either even shorter or can be scheduled in the dynamic segment. The distribution of the period and the resulting bandwidth is shown in Table I.

It should be up to the system designer, to choose the kind of free bandwidth. He may choose between free available bandwidth in used slots, if he expects a lot of integration of new functions in existing ECUs. Or it may be freely available slots, if he expects new ECUs at the bus or wants to be able to assign the slots for available ECUs freely. Additionally, he must take into account whether more static slots (which avoid jitter) or a larger dynamic segment (where the potential of less bus utilization due to shorter overheads) is more valuable.

Our scheduling tool can leave this decision open. If the minimum number of used slots is known (for a given payload, of course) and the minimum length of the dynamic segment is known, the border between the static and the dynamic segment can be done by an “engineered guess”.

Therefore, we provide a schedule for all possible static payloads. Afterwards, four numbers are given: (1) Bandwidth according to Equation 9, (2) Usage of used slots in percent, (3) maximum number of free static slots (if the dynamic segment is at its minimum size), and (4) number of possible static slots for in-cycle repetition. The last number is useful, if one expects messages which must be scheduled in half of the cycle time, i.e., 2.5 ms PDUs for a 5 ms cycle. That leads to a static segment, which is longer than 2.5 ms of course.

The result for some static payloads is given in Figure 4. The optimal schedule with respect to Equation 9 has a static payload of 58 Bytes, due to

TABLE I
DISTRIBUTION OF PDU PERIOD

Period	# PDUs	kBit/s
5	30	176.0
10	35	218.4
20	78	538.8
40	63	108.0
80	36	27.9
160	4	1.6
320	18	3.4
Sum	264	1074.1

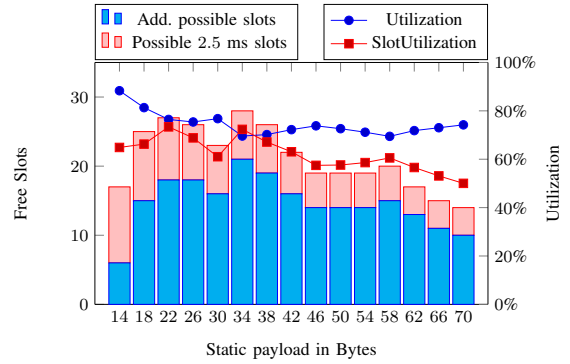


Fig. 4. Scheduling result for the case study.

a total utilization of 69.49%. (A payload of 34 bytes leads to a utilization of 69.59%. Additionally, also the utilized slots would leave more space for extensions, but only for the nodes which occupy them already)

The execution time of the proposed scheduling approach without employing the global optimization loop, see Figure 3, was 8 s on an Intel Core 2 Duo desktop PC with 2.4 GHz. As a reference, the best found solution as presented previously with 58 Bytes and 69.46 % utilization is used to investigate the efficiency of the global optimization loop. As a meta-heuristic optimization approach, simulated annealing was applied. The heuristic required an execution time of 19 minutes to find a solution of equal quality (in fact the same) as the proposed overall design flow without the global optimization. After 411 minutes, the optimization achieved an improvement of the utilization of only 0.2 %.

The case study indicates that the proposed overall design flow with its initial partitions based on rules of thumb may achieve very high quality solutions immediately. Given an execution time of only 8 seconds compared to several hours for an insignificant improvement of only 0.2% bus load, the global optimization should only be applied for the final system implementation. On the other hand, the 8 second execution time enables an interactive development and the exploration of several possible system architectures and, thus, provides a significant amount of automation and support for the automotive system engineer.

VI. CONCLUSION AND FUTURE WORK

We showed that a holistic FlexRay scheduling approach requires a deep investigation of the parameter set. FlexRay is highly configurable, but this is, with respect to the schedule, its flaw. Once chosen, the parameter set must not be changed during the lifetime of a car or even a platform of cars. Thus, the first schedule applied should be optimal, leaving enough space for future extensions of the system. In this work, we combined the parametrization with the scheduling of the static and the dynamic segment, due to their strong interdependence. The provided case studies gives evidence that the proposed holistic approach delivers better results than previous work.

REFERENCES

- [1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Syst. J.*, 4:25–30, March 1965.
- [2] S. Ding, N. Murakami, H. Tomiyama, and H. Takada. A ga-based scheduling method for flexray systems. *EMSOFT*, September 2005.
- [3] FlexRay Consortium. <http://www.flexray.com/>.
- [4] FlexRay Consortium. *FlexRay Communications System - Protocol Specification Version 2.1 Revision A*, December 2005.
- [5] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
- [6] M. Lukaszewicz, M. Głaż, P. Milbredt, and J. Teich. FlexRay Schedule Optimization of the Static Segment. In *Proceedings of CODES+ISSS*, pages 363–372, 2009.
- [7] I. Park. Flexray network parameter optimization method for automotive applications. *Industrial Electronics IEEE Transactions*, 58(4):1449–1459, 2010.
- [8] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the flexray communication protocol. *Real-Time Syst.*, 39:205–235, August 2008.
- [9] E. G. Schmidt and K. Schmidt. Message scheduling for the flexray protocol: The dynamic segment. *IEEE Transactions on Vehicular Technology*, 58(5), 2009.
- [10] K. Schmidt and E. G. Schmidt. Message scheduling for the flexray protocol: The static segment. *IEEE Transactions on Vehicular Technology*, 58(5), 2009.
- [11] H. Zeng, A. Ghosal, and M. Di Natale. Timing analysis and optimization of flexray dynamic segment. *CIT, 1932-1939 (2010)*, 2010.
- [12] H. Zeng, W. Zheng, M. D. Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli. Scheduling the flexray bus using optimization techniques. *Proceedings of DAC*, 2009.