Optimized Schedule Synthesis under Real-Time Constraints for the Dynamic Segment of FlexRay

Reinhard Schneider*, Unmesh Bordoloi†, Dip Goswami*, Samarjit Chakraborty*
*TU Munich, Germany, †Linkoeping University, Sweden

***Abstract*— The design process for automotive electronics is an iterative process, where new components and distributed applications are added over several design cycles incrementally. Hence, at each design iteration an existing communication schedule is extended by new messages that have to be scheduled appropriately. In this paper, the goal has been to synthesize schedules under real-time constraints for the *dynamic* segment of FlexRay with respect to the *64-cycle protocol* specification. We propose a flexible scheduling framework to generate all feasible schedules for a set of messages satisfying real-time and protocol constraints. Further, we present an optimization procedure to retain schedules according to suitable design metrics. Even though the size of the possible design space is exponential in the number of messages, our proposed method keeps down the schedule synthesis time to practically acceptable values as shown in the experiments.**

I. Introduction

Today, cars are becoming complex distributed embedded systems with a proliferation in the number of microcontrollers, sensors, actuators etc., which communicate over a fieldbus. The design process of the electronic components of such cars is an iterative process, where new features and components are added at each design cycle, the system is tested and validated and then the process is repeated [11], [12]. At each design cycle, new distributed applications might be added to the system which implies the addition of new tasks to existing Electronic Control Units (ECUs) and new messages to the existing communication network.

While scheduling such new messages, the goal is to satisfy all the protocol as well as timing constraints. Moreover, the schedules must be optimized according to specific design objectives, e.g., schedules should be such that messages in future iterations can easily fit into the system without having to change existing schedules. In this work, our goal is to synthesize schedules for the FlexRay protocol. We focus on FlexRay because it is emerging as the *de-facto* protocol for next generation cars, e.g., BMW has rolled out its 7 series with a FlexRay-equipped brake-by-wire application [2]. As the cost associated with FlexRay deployment is expected to go down over the next few years, more x-by-wire applications are expected to communicate over the FlexRay bus. Consequently, efficient FlexRay schedule synthesis techniques will become an ever-increasing challenge for future automotive applications.

Our contributions: FlexRay is a *hybrid* communication protocol for automotive networks, i.e., it allows the sharing of the bus between both time-triggered and event-triggered messages, offering the advantages of highly predictable temporal behavior and efficient communication bandwidth usage.
In FlexRay, the time-triggered component is the static (ST) segment and the event-triggered component is known as the dynamic (DYN) segment. Because of the inherent difficulty in analyzing the DYN segment, research has mostly focused on the ST segment. In fact, researchers have also focused on using the ST segment for incremental design, i.e., the problem of adding new messages to existing schedules [12]. On the other hand, the DYN segment being an event-triggered paradigm, actually offers more flexibility for such incremental design. This is because the communication slots of the ST segment are of fixed, and equal length, and are determined during design time. Thus, unexpected message scheduling requirements, which may come up during later stages of development might not fit into the ST segment slots. Further, certain application classes like those which demand high and variable data volumes may require a FlexRay network that mostly consist of the DYN segment. Such advantages offered by the DYN segment have, unfortunately, not been tapped yet. In light of the above facts, techniques to schedule messages on the DYN segment have high significance, and in this paper, we propose a mechanism for synthesizing such schedules with a focus on real-time guarantees and design flexibility.

Our results are especially interesting because the scheduling techniques proposed in this paper account for practically relevant details of FlexRay that have not been modeled in the existing literature on analysis for the FlexRay dynamic segment [5], [9]. In particular, our proposed technique synthesizes schedules from the perspective of the *64-cycle matrix*, which allows *multiplexing*, i.e., multiple messages can be assigned the same priority (slot) if they are mapped to the different cycles. The details of the *64-cycle matrix* will be described in Section II. Further, we synthesize and optimize the schedules for a set of messages which makes our results meaningful from a practical perspective.

Overview of our scheme: Our proposed scheme (illustrated in Fig. 1) is an incremental schedule synthesis engine that we run for every *new* application that is to be mapped on the FlexRay DYN segment. As an input to the scheduler we consider (i) the *existing* schedules that have been generated at previous design cycles, (ii) the FlexRay network configuration and (iii) the *set of new* messages, that is to be scheduled at the current design cycle. Based on this input specification we synthesize the schedules for the *set of new* messages by applying several pruning techniques to discard *infeasible* solutions.
At the first stage, our scheme prunes schedules that do not

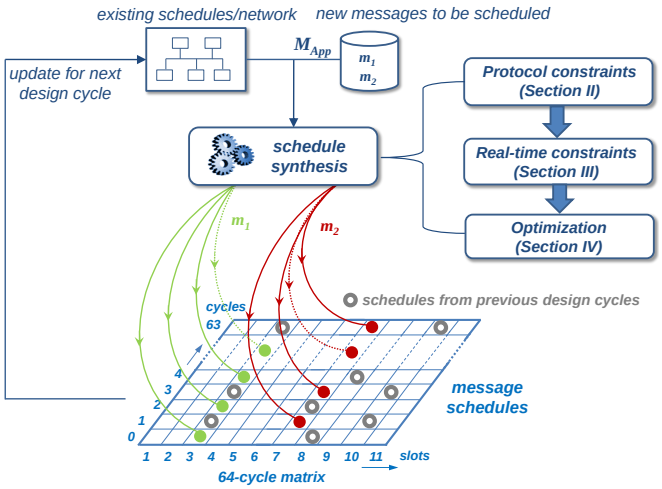

Fig. 1. Overview of our incremental schedule synthesis engine.

satisfy the FlexRay protocol specification (see Section II). At the second stage (discussed in Section III), schedules violating specified real-time constraints are filtered out. This pruning is done based on algebraic computations from which we derive certain bounds on the scheduling parameters that may be assigned to each *new* message. Hence, the *feasible* design space can dramatically be restricted. Next, we apply a timing model to check if the generated schedules satisfy given message deadlines. Schedules where messages violate the deadlines are discarded.

As the number of possible schedules might be exponential in the number of *new* messages, schedule synthesis is a computationally expensive problem. However, by applying effective pruning strategies in both the above stages, a large portion of the design space is discarded, which makes the problem tractable and the approach efficient in practice.

Finally, all remaining schedules, i.e., those which satisfy the FlexRay protocol and specified real-time constraints are optimized according to multiple design objectives (see Section IV). Here, a system designer may choose suitable weights on several objectives that are relevant to his or her design requirements, and retain the optimal schedules for all *new* messages. In Section V we apply our scheme to several use cases in order to show the applicability of our approach.

Fig. 1 illustrates our scheme. The three core components of our scheduling engine, i.e., pruning based on *protocol* and *real-time constraints* followed by an *optimization* procedure are shown here. Further, it illustrates that *new* message schedules generated in the current design cycle will become a part of the input specification for the next design cycle. This figure depicts the synthesized FlexRay schedules in the form of a matrix, where two *new* messages m_1 and m_2 are being added by allocating different cycles in slots 3 and 7.

Related work: In recent years, there has been a tremendous interest in timing and schedulability analysis for the FlexRay protocol. Techniques for scheduling messages in the static segment of FlexRay have been described in [6], [12]. Methods have been proposed in [7], [12] specifically to generate extensible schedules suitable for the automotive electronics design cycle. It may be noted that [6] generates schedules considering

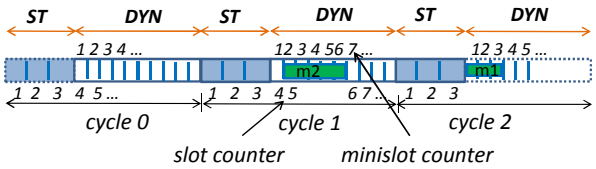

Fig. 2. Transmission over the DYN segment.

the 64-cycle matrix of the FlexRay protocol. However, all of the above attempts ignored the DYN segment, which is the focus of this paper.

Recently, there has been some research effort towards analyzing the timing properties of the DYN segment of FlexRay. Notable among these are [5], [9]; but all of them assumed that message schedules are *given* and tried to estimate the worst case response time of the messages. In this paper, we propose a framework to *synthesize* schedules for a set of messages in the DYN segment of FlexRay.

The only known attempts to generate message schedules on the FlexRay DYN segment were reported in [8] and [10]. In [10], the scheduling problem has been formulated as a nonlinear integer programming problem and the objective was to minimize the required size of the DYN segment. In [8], both the ST and the DYN segment have been considered together. The problem of assigning priorities to messages in the DYN segment has been solved using a heuristic which optimizes the schedule by maximizing the difference between the message delay and its deadline. Although both papers are interesting approaches towards schedule synthesis for the FlexRay DYN segment, there are some drawbacks which might hinder their applicability in practice.

The first drawback being that none of them consider the iterative design process of the automotive electronics industry. Thus, schedules generated are not optimized for *flexibility*. Secondly, [8], [10] made certain simplifying abstractions on the FlexRay protocol to make the analysis tractable. In particular, the 64-cycle matrix and protocol restrictions like *pLatestTx* (which is described later) were not modeled. Clearly, such abstractions restrict the applicability of these techniques in practice. This paper addresses the challenge to synthesize optimized message schedules under real-time constraints considering multiple design objectives.

II. The FlexRay Protocol

The FlexRay Communication protocol [4] is organized as a *periodic sequence* (known as the 64-cycle matrix) of communication cycles, where each cycle is of a fixed length, denoted by *gdCycle*. As discussed in Section I, each communication cycle is further subdivided into two major segments: ST and DYN. Furthermore, the FlexRay communication cycle also contains a mandatory Network Idle Time (NIT) segment that is used to perform clock synchronization. We will ignore the NIT segment in further discussions because it does not interfere with the DYN segment and hence, its blocking time may easily be account for. In the following we discuss the protocol specification centering around the DYN segment. First, we discuss the DYN segment in the context of one communication cycle, followed by its description in the context of the of the

64-cycle matrix.

Dynamic segment: The DYN segment is partitioned into equal-length slots, which are referred to as "minislots". Each minislot is numbered by a *minislot counter*. The minislot counter counts the number of minislots from the beginning of the DYN segment, as shown in Fig. 2. Messages on the DYN segment are assigned fixed priorities. At the beginning of each DYN segment, i.e., when the minislot counter is one, the highest priority message is allowed to be transmitted by occupying the required number of minislots. However, if the message is not ready, then only one minislot is consumed. In either case, the bus is then given to the next highest-priority message and the same process is repeated until the end of the DYN segment. Fig. 2 shows an illustrative example for three cycles. Let m_1 be the highest priority message, which must get access to the bus at the start of DYN segment. Thus, if m_1 misses its turn, as shown in cycle 1 of the figure, one minislot is wasted, after which the next higher priority message, m_2 is allowed access to the bus in cycle 1. In cycle 0 no messages are ready.

Each minislot, where a message may be transmitted, is numbered by the *slot counter*. The set of static slots is denoted by $S_{ST} = \{1, ..., l\}$. If there are l number of ST slots, the slot counter in the DYN segment starts at $l+1$. The set of dynamic slots is given by $S_{DYN} = \{l+1, ..., n\}$. Hence, in Fig. 2 the slot counter for DYN segment starts from 4 (see cycle 0). When a message transmission occupies multiple minislots, the slot counter is increased only by one. For example, in cycle 1 of Fig. 2, m_2's transmission starts in minislot 2 and occupies 5 successive minislots. Thus, the minislot counter changes from 2 to 7 while the *slot counter* changes from 5 to 6.

Each message m_i is assigned a slot number S_i, which specifies that the message may be transmitted in the DYN segment when S_i is equal to the current value of *slot counter*. Messages having a higher priority are assigned lower slot numbers so that they have access to the bus first. For example, in Fig. 2, m_1 is assigned slot number $S_1 = 4$ and m_2 is assigned slot number $S_2 = 5$. In cycle 1, m_2 gets access to the bus when the *slot counter* becomes 5 after m_1 misses its turn when *slot counter* was 4.

Note that when its turn comes, a message is transmitted only if the current minislot counter value is not greater than *pLatestTx* which denotes the highest minislot counter value a message transmission is allowed to begin for a certain ECU. The value of *pLatestTx* is statically configured during design time and depends on the maximum dynamic payload size that is allowed to be transmitted by a certain ECU (please see [4]).

64-cycle matrix: In the above we described the DYN segment which appears in each communication cycle. However, a set of 64 such communication cycles is repeated in a periodic sequence where each cycle is indexed by a *cycle counter*. The cycle counter is incremented from 0 to 63 after which it is reset to 0. Consequently, 64 consecutive communication cycles constitute a communication pattern which is repeated.

To identify the actual transmission cycles within the 64-cycle matrix two parameters are introduced: (i) the base cycle B_i which denotes the offset within 64 communication cycles, and (ii) the cycle repetition rate R_i, which indicates the number of cycles that must elapse between two consecutive allowable transmissions. We already discussed the slot number S_i associated with each message m_i, which denotes the communication slot within a cycle where the message will be transmitted. Thus, any FlexRay message m_i is assigned S_i, B_i, and R_i to uniquely specify admissible transmission points within 64 cycles. In the rest of the paper we refer to $\Theta_i = \{S_i, B_i, R_i\}$ as the *schedule* of m_i. Further, we refer to the set of *admissible cycles* Γ_i as the *cycle counter* of Θ_i.

According to the FlexRay communication protocol ([1], [4]), the following relations must hold among these parameters for any schedule Θ_i:

FR1) $B_i \in \{0, ..., 63\}$.
FR2) $R_i = 2^r$; r $\in \{0, ..., 6\}$.
FR3) $B_i < R_i$.
FR4) $\gamma_n = (B_i + n \cdot R_i) \mod 64$, $n \in \{0, 1, 2, ...\}$ and $\gamma_n \in \Gamma_i$.

It may be noted that when $R_i = 1$, B_i must be 0 due to FR3. This setting implies that the message m_i is allowed to be transmitted in every cycle.

Illustrative examples: We will show two example schedules to illustrate the 64-cycle scheduling mechanism. In the first example (Fig. 3(a)), the ST segment is configured with 3 slots and the DYN segment consists of 10 minislots. Let m_1, m_2 and m_3 be three DYN segment messages with associated schedules $\Theta_1 = \{7, 0, 2\}$, $\Theta_2 = \{9, 1, 2\}$, $\Theta_3 = \{11, 0, 4\}$. It should be pointed out that the marked cells in the 64-cycle matrix show minislots in certain cycles where message transmissions *may* begin. If a message is not ready for transmission, a minislot goes empty as explained earlier. On the other hand, if a message is ready, it will occupy the required number of minislots in the same cycle to complete its transmission.

In the second example (Fig. 3(b)), we have considered ST and DYN segments similar to the previous example with messages m_1, m_2, m_3 and m_4 being mapped to the DYN segment. The DYN segment messages have the schedules $\Theta_1 = \{7, 0, 2\}$, $\Theta_2 = \{7, 1, 2\}$, $\Theta_3 = \{11, 0, 4\}$, $\Theta_4 = \{11, 1, 2\}$. Here, identical slot numbers S_i have been chosen for messages m_1, m_2 and m_3, m_4 with non-identical values of R_i and B_i. Such scheduling leads to *slot multiplexing*, i.e., the same slot is being used by multiple messages in different cycles. For example, m_1 is assigned slot 7 in the even cycles and m_2 in the odd cycles. *Slot multiplexing* improves the bandwidth utilization as the resources (cycles) of one slot are distributed to several messages. Thus, for any messages m_i and m_j with schedules Θ_i and Θ_j which share the same slot, the following condition holds:

$$\Gamma_i \cap \Gamma_j = \emptyset, \; S_j = S_i. \tag{1}$$

Problem formulation: In this paper we propose a technique to synthesize message schedules for the FlexRay DYN segment. We denote M_{App} as the set of *new* application messages that are to be scheduled on top of an existing network. Every such message $m_i \in M_{App}$ is specified by the 3-tuple $(c_i, \; d_i, \; p_i)$

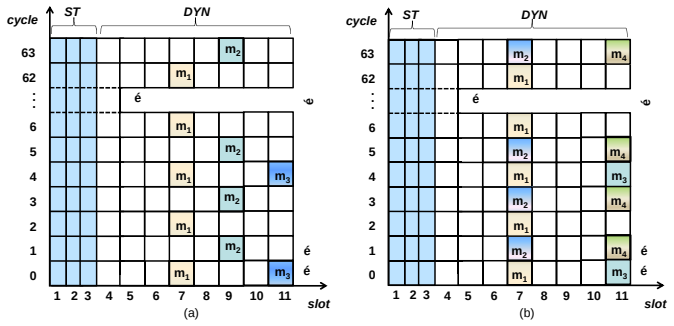

Fig. 3. The 64-cycle matrices for two examples.

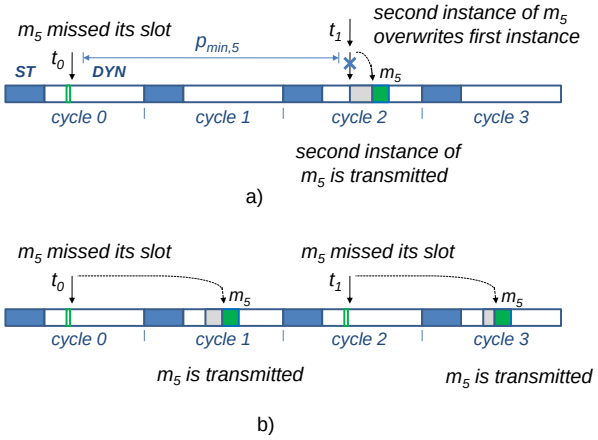

Fig. 4. Scenarios for message transmission with different repetition rates a) $R_5 = 2$, b) $R_5 = 1$.

where c_i denotes the message size in terms of minislots, d_i indicates the message deadline and p_i is the minimum inter-arrival distance between two consecutive data items arriving from a task that will be packetized as a FlexRay message. The goal is to synthesize all feasible schedule solutions $\omega_k \in \Omega$ with $\omega_k = \{\Theta_1, \Theta_2, ..., \Theta_n\}$ for every $m_i \in M_{App}$ that satisfy the FlexRay specification and specified real-time constraints without changing the existing schedules that have been synthesized at previous design iterations. Further, we optimize the set of *feasible* schedules Ω for the optimal $\omega_{opt,k} \in \Omega$ according to multiple weighted design objectives.

III. Real-Time Constraints

As discussed in the previous section, our goal is to synthesize message schedules such that the hard real-time constraints of the messages are satisfied, i.e., delays suffered by the messages must not exceed their deadlines. The two key parameters in a message schedule that affect message delays are the repetition rate R_i and the slot S_i. On the one hand, R_i determines how many communication cycles a message will have to wait in case it missed its slot in a cycle. On the other hand, the slot S_i determines the priority of a message to access the bus within the present cycle. Hence, the choice of both these parameters are crucial to satisfy real-time constraints. In the following, we discuss how we may compute *necessary upper bounds* on these parameters for each message m_i such that the message delays are safely bounded. There might still be many admissible values of R_i and S_i for a message m_i within these bounds. However, these bounds might not be *sufficient bounds* with respect to the message deadlines, i.e., m_i might miss its deadline for any Θ_i with $R_i \leq R_{max,i}$ and $S_i \leq S_{max}$. Hence, we will also present a *delay model* to compute the message delay for any schedule Θ_i of m_i, based on which we may discard choices of schedules which do not satisfy the real-time constraints.

A. Computing bounds on repetition rate

Given a repetition rate R_i, the FlexRay bus potentially provides the assigned slot S_i to transmit a message m_i every R_i cycles, i.e., within a time interval of $R_i \cdot gdCycle$. We define, the sampling period of the FlexRay bus for message m_i as the interval length $R_i \cdot gdCycle$. Note that this sampling rate of the FlexRay bus for m_i has to satisfy the minimum inter-arrival distance p_i of two consecutive data items processed by the sender task. This problem is easily solved if the local ECU clocks are synchronized to the slots in the FlexRay bus, like in the ST segment. In such a scenario, $p_i = R_i \cdot gdCycle$ is a sufficient condition in order to successfully transmit every instance of m_i. In contrast to the ST segment, the location of the communication slots in the DYN segment can vary from cycle to cycle as it depends on the actual transmission of messages that are assigned higher priorities, i.e., smaller slot numbers (see Section II). Thus, in the DYN segment, the precise time at which a message is ready for transmission can be before or after the corresponding dynamic communication slot was ready. Consequently, a bus sampling rate equal to the maximum message rate does not provide safe transmission points. In the worst case, data can not be transmitted on the bus before a new value is produced by the application as illustrated in Fig. 4(a). Message m_5 has a repetition rate of $R_5 = 2$. The minimum sender task period p_5 is defined by $p_5 = 2 \cdot gdCycle$. In cycle 0, m_5 just missed its slot S_5 at t_0 and at $t_1 = t_0 + p_i$ the next slot S_5 is not yet ready as messages having higher priority than m_5 are transmitted first. Thus, the first instance of m_5 is overwritten by the second instance of m_5 which finally gets transmitted on the bus in cycle 2.

To reflect the sampling constraints in the DYN segment, we require that the FlexRay sampling rate has to be at least twice the message activation rate. This is illustrated in Fig. 4(b). The message m_5 is now assigned a repetition rate of 1 and can therefore safely be transmitted on the bus before the next instance of m_5 is computed and written to the transmit buffer of the FlexRay controller. Formally, we obtain the following mathematical inequalities:

$$R_i \leq \frac{p_i}{2gdCycle} \tag{2}$$

As the repetition rate is only defined by a power of two and is limited to 64 (see FR2 in Section II), the maximum repetition rate is computed as:

$$R_{max,i} = \min(2^{\lfloor \log_2(\frac{p_i}{2gdCycle}) \rfloor}, 64) \tag{3}$$

Note that according to FR3, we have $B_{max,i} < R_{max,i}$ for the maximum base cycle $B_{max,i}$.

B. Computing bounds on slot range

As in the case of the repetition rate, we will now derive bounds on the admissible slot range. Note that a message m_i in the DYN segment can get delayed by several cycles if the *min-islot counter* exceeds the specified $pLatestTx$ value. In that case m_i is *displaced* because of interference from messages having higher priorities. In such scenarios, m_i is delayed to the next feasible communication cycle $\gamma_n \in \Gamma_i$, where it may get *displaced* once again. Thus, the displacement of a message in the DYN segment may lead to unbounded message delays. In this work, we are interested in synthesizing message schedules where the delays are safely bounded and satisfy hard real-time constraints. Hence, our goal is to synthesize message schedules that allocate only those slots S_i where message transmissions are guaranteed without the risk of displacement. Towards this, we compute a slot called S_{max}, which is the last slot in the DYN segment that may be assigned to any message. By assigning slot numbers $S_i \leq S_{max}$, the schedule guarantees message transmission without any displacement, i.e., the delay is safely bounded. As the displacement of m_i depends on the transmission of existing messages having a higher priority we introduce a matrix A that accounts for the slots and cycles that are allocated by such existing schedules. Let A be a $n \times m$ system matrix where $n = 64$ and $m = |S_{DYN}|$. $A_{i,j}$ indicates whether cycle $i \in \{0, 1, 2, ..., 63\}$ of slot $j \in S_{DYN}$ is already allocated by an existing message schedule:

$$A_{i,j} = \begin{cases} 0, \text{if } \textit{not allocated} \\ 1, \text{if } \textit{allocated by a schedule} \end{cases} \quad (4)$$

Algorithm 1 Computing the bound on the slot range.

Input: System matrix A

1: $w_i = 0, \forall i \in \{0, ..., 63\}$
2: **for all** $i \in \{0, ..., 63\}$ **do**
3: **for all** $j \in S_{DYN}$ **do**
4: **if** $A_{i,j}$=1 **then**
5: $w_i = w_i + c_{i,j}$
6: **else**
7: $w_i = w_i + c_{max}$
8: **end if**
9: **if** $w_i \leq \min(pLatestTx)$ **then**
10: $S_{max,i} = j$
11: **end if**
12: **end for**
13: **end for**
14: $S_{max} = \min S_{max,i}$

Given A, S_{max} is computed according to Alg. 1. We iterate over all cycles i in every slot $j \in S_{DYN}$ (lines 2 and 3) and check if the total workload $w(i)$ per cycle exceeds the minimum $pLatestTx$ value of all nodes ECUs in the network (line 9). Towards this we sum up the maximum possible workload per slot for every cycle. For $A_{i,j} = 1$ the maximum resource consumption in terms of minislots is denoted by the message size $c_{i,j}$ that may be consumed in cycle i in slot j (line 5). If $A_{i,j} = 0$ (line 7) naturally one minislot is consumed. However, to ensure that *future* messages may allocate this empty slot, we specify a maximum admissible message size, i.e., $c_{max} = c(maxPayload)$ where the value of $maxPayload \leq 254bytes$ is a design choice of the system designer. Finally, we take the minimum S_{max} among all cycles

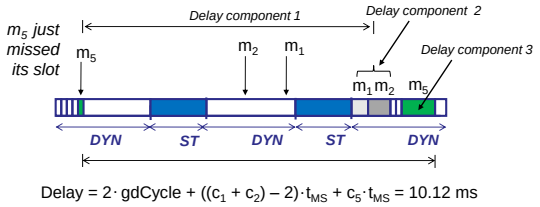

Fig. 5. The worst case delay scenario for m_5.

(line 14) to obtain a safe bound on the slot range for the 64-cycle matrix. As Alg. 1 illustrates the computation of S_{max} also exploits on the *existing* message sizes. Thus, the value S_{max} can increase, i.e., the slot range may improve over several design iterations because the actual sizes of messages added in previous iterations might be smaller than c_{max} and therefore relax the computation of the total workload $w(i)$.

C. Delay model

In this section we derived bounds on the slot range S_{max} and the maximum repetition rate $R_{max,i}$ such that message delays are safely bounded. Using these bounds we are able to significantly restrict the design space for the set of admissible schedules. Moreover, there might be many different schedules Θ_i within S_{max} and $R_{max,i}$ that might not satisfy specified real-time constraints. This is because the worst case delay D_i of m_i in the DYN segment depends on the higher priority messages as well, which might interfere with m_i leading to deadline violations. In order to discard schedules which might violate deadlines, we propose a timing model to compute the worst-case delay D_i of a message m_i. If $D_i > d_i$, where d_i is the deadline of m_i, the schedule does not satisfy the real-time constraints and will therefore be discarded.

In the following, we will utilize a running example to explain the proposed timing model. Towards this, consider a FlexRay configuration with $gdCycle = 5ms$ that is divided into a ST segment of $2ms$ and a DYN segment of $3ms$. Let the number of ST segment slots be $|S_{ST}| = 10$ and the number of minislots per DYN segment $n_{MS} = 300$. The duration of one minislot is considered as $t_{MS} = 0.01ms$. We consider six messages being transmitted over the DYN segment with schedules $\Theta_1 = \{11, 0, 4\}$, $\Theta_2 = \{12, 0, 2\}$, $\Theta_3 = \{11, 1, 2\}$, $\Theta_4 = \{13, 1, 4\}$, $\Theta_5 = \{14, 0, 2\}$ and $\Theta_6 = \{15, 0, 1\}$. Message priorities are determined according to the slot numbers S_i in descending order, i.e., $S_1 = 11$ denotes highest priority. Message sizes in minislots are $c_1 = c_3 = 3$, $c_2 = c_6 = 5$, $c_4 = 4$ and $c_5 = 6$.

The worst-case delay experienced by any message transmitted in the DYN segment is made up of three delay components. The *first* delay component captures the bus blocking time due to the repetition rate R_i. If m_i arrives just after the beginning of the slot S_i, then it has to wait for its next slot after R_i cycles, e.g, if m_5 arrives just after slot 14 in any of the cycles $\gamma_n \in \Gamma_5$, it has to at least wait for 2 cycles, i.e., $2 \cdot gdCycle$ = 10 ms for its next slot S_5. Therefore, the bus blocking time accounts for $R_i \cdot gdCycle$ (see Fig. 5).

The *second* delay component is contributed by messages having higher priority than m_i that are transmitted in the

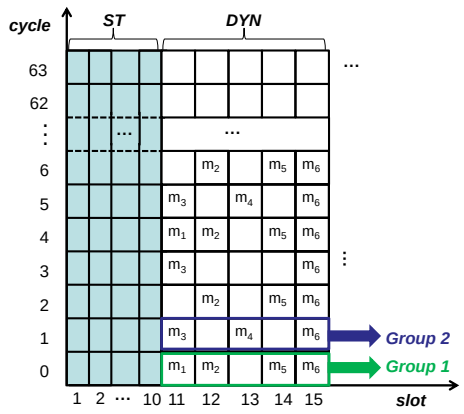

Fig. 6. *Groups* of the DYN segment messages.

same cycle. In other words, the transmission of another DYN segment message m_j can affect the delay of m_i, if the priority of m_j is higher than m_i's as well as the cycle counters of m_i and m_j have at least one common element, i.e., $\Gamma_i \cap \Gamma_j \neq \emptyset$. In our example, the transmission of m_5 is not influenced by m_6 because m_6 has lower priority than m_5. Messages m_i, $i \in \{1, 2, 3, 4\}$ have higher priority than m_5. However, as $\Gamma_1 \cap \Gamma_2 \cap \Gamma_5 \neq \emptyset$ and $\Gamma_3 \cap \Gamma_4 \cap \Gamma_5 = \emptyset$, only m_1 and m_2 can affect the delay of m_5 (see Fig. 5). To formalize the above idea, we divide the messages into *groups*. A *group* $G_k = \{i, j, ...\}$ is a set of message indices such that $\Gamma_i \cap \Gamma_j \cap ... \neq \emptyset$. There are two groups in the above example, i.e., $G_1 = \{1, 2, 5, 6\}$ and $G_2 = \{3, 4, 6\}$ as $\Gamma_1 \cap \Gamma_2 \cap \Gamma_5 \cap \Gamma_6 \neq \emptyset$ and $\Gamma_3 \cap \Gamma_4 \cap \Gamma_6 \neq \emptyset$ (Fig. 6). Message m_5 belongs to *group* G_1 and the messages m_1 and m_2 have higher priorities than m_5. In the worst case scenario, both m_1 and m_2 are transmitted before the transmission of m_5. The additional delay is computed as $(c_1 - 1 + c_2 - 1) \cdot t_{MS} = 0.06\ ms$ (note, that one minislot is consumed even if no message is being transmitted). Further, certain messages can belong to more than one group. For example, m_6 is in both G_1 and G_2. For m_6, the delay due to the transmission of higher priority messages in G_1, i.e., m_1, m_2 and m_5 is $((c_1 + c_2 + c_5) - 3) \cdot t_{MS} = 0.11\ ms$. Similarly, for G_2 we obtain $((c_3 + c_4) - 2) \cdot t_{MS} = 0.05\ ms$. Therefore, m_6 experiences the worst case delay in G_1. The delay experienced by any message m_i due to the transmission of messages having higher priority is given by:

$$\max_{\forall k} \sum_{j} (c_j - 1) \cdot t_{MS}, \forall j \in G_k \ s.t.\ S_j < S_i. \quad (5)$$

The *third* component in the delay value is the communication time of the message. For example, the communication time of m_5 is $c_5 \cdot t_{MS}$, i.e., $0.06\ ms$. Therefore, the overall worst case delay for m_5 = $(10 + 0.06 + 0.06)\ ms$ = $10.12\ ms$ (Fig. 5). The total delay experienced by any message m_i that is mapped on the DYN segment with corresponding schedule $\Theta_i = \{S_i, B_i, R_i\}$ is computed as:

$$D_i = R_i \cdot gdCycle + \max_{\forall k} \sum_{j} (c_j - 1) \cdot t_{MS} + c_i \cdot t_{MS}, \quad (6)$$
$$\forall j \in G_k \ s.t.\ S_j < S_i.$$

Now, we are ready to present our algorithm (Alg. 2) to synthesize all possible schedules (denoted by the set Ω) which satisfy: (i) the FlexRay protocol constraints (Section II), and (ii) the real-time constraints (described in this section). Therefore, we generate the schedules bounded by S_{max} and

Algorithm 2 Schedule synthesis under protocol and real-time constraints.

Input: Existing schedules Θ_j, messages to be scheduled $m_i \in M_{App}$, deadlines d_i

```
1:  Ω = {}
2:  for all S_i ≤ S_max do
3:    for all R_i ≤ R_max,i do
4:      for all B_i < R_i do
5:        if (S_i ≠ S_j) ∨ (Γ_i ∩ Γ_j = ∅), ∀i, j then
6:          D_i = computeDelay()
7:          if D_i ≤ d_i, ∀i ∈ {1, ..., n} then
8:            ω = {Θ_1, ..., Θ_n}
9:            Add ω to set of feasible schedules Ω
10:         end if
11:       end if
12:     end for
13:   end for
14: end for
```

$R_{max,i}$ that satisfy the protocol constraints according to FR1 to FR4 (lines 2 to 4) and Eq. 1 (line 5). Next, we compute message delays according to the proposed delay model (line 6) for every $m_i \in M_{App}$. Only if *all* messages meet their deadlines we retain the *feasible* solution $\omega_k \in \Omega$ (lines 7 to 9) otherwise the solution is discarded.

IV. Optimization

In the previous section, we discussed how to synthesize a set of schedules for *new* messages $m_i \in M_{App}$ on top of *existing* schedules. However, there can be potentially many different solutions from Alg. 2 that are valid with respect to the protocol and specified real-time constraints. Given the set of *feasible* schedules Ω, the challenge is to determine the optimal $\omega_k \in \Omega$ according to specific design objectives. On the one hand, each schedule Θ_i should guarantee specified real-time constraints during the entire development cycle, e.g., messages having a high priority that may be accommodated in the future should not cause deadline violations of existing messages having a lower priority. In order to meet this design requirement a schedule shall provide sufficient slack, i.e., the schedule Θ_i should be such that a message m_i can tolerate additional delay caused by interference of *future* messages having higher priorities. On the other hand, the network schedule should provide sufficient available slots and cycles in order to accommodate additional *future* messages satisfying their specified real-time constraints. Consequently, the optimization problem according to multiple, potentially contradictory, design objectives is a major challenge we want to address in what follows.

Schedule optimization: In the following we outline an optimization procedure to determine the optimal $\omega_{opt,k} = \{\Theta_1, ..., \Theta_n\}$ from the set of *feasible* schedules Ω. Further, our procedure allows the system designer to balance the different metrics according to his or her design needs. First, we quantify the total number of dynamic communication slots where the delay is safely bounded by $S_{max} - l$ with l denoting the number of static slots. Further, the higher the slot number S_i

that is assigned to a *new* message m_i the more slots with small numbers denoting high priorities might be available in order to schedule *future* messages, i.e., the flexibility and extensibility for the synthesis of *future* schedules is improved. Second, we want to account for the available communication cycles in the 64-cycle matrix. Recall from FR2 that within an empty slot S_i, a message m_i can be assigned any repetition rate $R_i = 2^r$ with r $\in \{0, ..., 6\}$, i.e., $R_i \in \{1, 2, 4, 8, 16, 32, 64\}$. For each of these repetition rates there are R_i base cycle values available as $B_i < R_i$ (see FR3). For example, with a repetition rate $R_i = 2$, m_i might have a base cycle $B_i \in \{0, 1\}$, with $R_i = 4$, m_i might have any base cycle $B_i \in \{0, 1, 2, 3\}$. Thus, the total number of choices to schedule m_i in an empty slot S_i is computed as $\sum_{i=1}^{|R|} R_i = 1+2+4+8+16+32+64 = 127$. Let m_1 be a message with $\Theta_1 = \{S_1, 0, 2\}$, we want to compute the number of available choices in S_1 in order to schedule a *future* message m_2 in the same slot. Consequently, the number of remaining choices in S_1 that are available for m_2 is computed as $0 + 1 + 2 + 4 + 8 + 16 + 32 = 63$, where each term in this sum captures the number of choices of each repetition rate R_i. Thus, the higher the repetition rate of m_i the more schedules are available for *future* messages in S_i.
Finally, the slack of a message m_i is defined as $s_i = d_i - D_i$ which is the difference between the message delay and its deadline. This means a smaller slack for m_i may lead to deadline violations if interfering messages having a higher priority are schedules in future. Note, that the slack not only depends on the slot number S_i but also on the interference due to higher priority messages in every cycle $\gamma_n \in \Gamma_i$ of S_i and the bus blocking time $R_i \cdot gdCycle$.
We define the objective Ψ_i for a message m_i as

$$\Psi_i = \lambda_1^i \left(\frac{S_i - l}{S_{max} - l} \right) + \lambda_2^i \left(\frac{R_i}{R_{max,i}} \right) + \lambda_3^i \left(\frac{s_i}{d_i - c_i \cdot t_{MS}} \right). \quad (7)$$

The weights λ_1 and λ_2 are suitably chosen by the system designer in order to account for the slot and cycle availability for *future* messages and λ_3 is chosen to weight the message slack. Each term in Eq. 7 is normalized by its maximum value. Overall, the optimal solution can be defined as the schedules $\omega_{opt,k} = \{\Theta_1, ..., \Theta_n\}$ that maximize the cost function

$$\Psi(\omega_k) = \sum_{i=1}^{n} \Psi_i. \quad (8)$$

Hence, we define the overall optimal cost $\Psi^*(\omega_{opt,k})$ as

$$\Psi^*(\omega_{opt,k}) = \max_{k=1,...,|\Omega|} \Psi(\omega_k). \quad (9)$$

V. Experimental results

In this section we illustrate the applicability of our proposed scheduling engine through detailed experimental results.

Experimental setup: The proposed scheduling engine has been implemented in Matlab. The experiments have been carried out on a XP machine with a dual core 1.8 GHz processor with 3 GB RAM. We used the Elektrobit Tresos Designer Pro 2009.a V4.4 [3] tool to determine the FlexRay bus configuration parameters. Thus, the values we obtained satisfied the protocol specifications and are practically relevant. The FlexRay cycle length was set to $5ms$ with the length of the DYN segment set to $3.615ms$. The rest of the FlexRay cycle was distributed between the ST segment of 17 static slots and the NIT segment. The number of minislots in one DYN segment was configured to 241 and each minislot duration was of $0.015ms$. The value of *maxPayload* was set to 128 bytes. To provide meaningful results we considered 14 existing schedules m_j, with $j \in \{1, ..., 14\}$, that represent legacy components covering basic system functionalities. Table I gives an overview of message schedules $\Theta_j = \{S_j, B_j, R_j\}$, payload sizes P_j denoted in bytes and message sizes c_j in minislots. For the purpose of illustrative examples, we considered a new application that consists of three messages $m_i \in M_{App}$, with $i \in \{15, 16, 17\}$, that is to be scheduled on top of the existing FlexRay network from Table I. The message properties are given by the deadlines d_i, the minimum inter-arrival periods p_i and the total message sizes in minislots c_i. We considered the following message properties: $p_{15} = 30ms$, $d_{15} = 30ms$, $c_{15} = 4$, $p_{16} = 50ms$; $d_{16} = 50ms$, $c_{16} = 5$ and $p_{17} = 50ms$, $d_{17} = 20ms$, $c_{17} = 3$.

Schedule synthesis and optimization: Using Eq. 3 the static bounds on the repetition rates have been computed as $R_{max,15} = 2$, $R_{max,16} = 4$ and $R_{max,17} = 4$. Further, using Alg. 1 we derived $S_{max} = 41$ considering the existing schedules from Table I as an input. Hence, the slot range where the delay is safely bounded is defined as $S^*_{DYN} = \{18, 19, ..., 41\}$. In total our scheduling engine evaluated 631,096 schedules where 384,682 solutions have been discarded due to deadline violations. Thus, the optimization procedure has been applied to $|\Omega| = 246,414$ feasible solutions in order to synthesize the optimal schedules. The total runtime was approximately 10 minutes. We demonstrate the applicability of our schedule synthesis engine by several illustrative use cases for different weights. For better readability we write λ_1, λ_2, λ_3 instead of λ_1^i, λ_2^i, λ_3^i with $i = \{15, 16, 17\}$. Table II shows the schedules Θ_{15}, Θ_{16}, Θ_{17} for the optimal cost $\Psi^*(\omega_{opt,k})$ for every use case. Note, that depending on the choice of $\lambda_1^i, \lambda_2^i, \lambda_3^i$ there may be more than one optimal cost $\Psi^*(\omega_{opt,k})$.

(Table II, case 1): Maximizing slot availability
Messages m_{15} and m_{17} have been *multiplexed* within slot 41. Hence, they share the highest admissible slot number $S_{15} = S_{17} = S_{max}$, i.e., the lowest available priority has been assigned to both the messages. As no more cycles are available in S_{max}, i.e., $\Gamma_{15} \cup \Gamma_{17} = \{0, 1, 2, ..., 63\}$, m_{16} has been assigned the second highest slot number $S_{16} = 40$. Consequently, slots with low slot numbers (high priorities) are available for *future* messages. As no constraint on the repetition rates has been specified (as $\lambda_2 = 0$), there are many available choices for $R_{16} \leq R_{max,16}$ and $B_{16} < R_{16}$.

(Table II, case 2): Maximizing cycle availability

TABLE I
EXISTING FLEXRAY SCHEDULES.

index j	schedule Θ_j	payload P_j in bytes	size c_j in minislots
1	$\{18, 0, 2\}$	8	3
2	$\{21, 1, 4\}$	8	3
3	$\{22, 0, 2\}$	8	3
4	$\{24, 2, 4\}$	10	3
5	$\{25, 0, 1\}$	12	3
6	$\{27, 0, 2\}$	8	3
7	$\{30, 0, 1\}$	6	2
8	$\{35, 3, 4\}$	6	2
9	$\{36, 0, 2\}$	8	3
10	$\{37, 1, 2\}$	6	2
11	$\{38, 0, 2\}$	8	3
12	$\{42, 1, 2\}$	16	4
13	$\{44, 0, 8\}$	8	3
14	$\{44, 1, 2\}$	8	3

As $\Psi^*(\omega_{opt,k})$ only depends on the repetition rates R_i we obtain a large set of optimal solutions. All messages are assigned their maximum feasible repetition rates for every slot $S_i \in S^*_{DYN}$ with respect to their deadline constraints, i.e., $R_{15} = R_{max,15}, R_{16} = R_{max,16}$, and $R_{17} = 2 < R_{max,17}$ as the deadline $d_{17} = 20ms$ will be violated for $R_{17} = R_{max,17} = 4$ (see Eq. 7).

(Table II, case 3): Maximizing slack
As already discussed in the previous section slack not only depends on the slot number S_i but also on the bus blocking time $R_i \cdot gdCycle$ as the worst-case delay experienced by any message m_i increases with R_i (see Eq. 7). Hence, for $R_i = 1$, the blocking time is minimal and the optimal schedules Θ_i are such that S_i *and* R_i are minimized. The drawback of a purely slack optimized schedule is the high reservation of unused bandwidth as one complete slot is reserved for every message m_i regardless of the actual message periods.

(Table II, case 4): Maximizing cycle availability and slack
If we equally weight both objectives we resolve the drawback of use case 2 and 3. High priority slots have been assigned while the repetition rates have been maximized, i.e., an efficient bandwidth utilization. Moreover, the slot assignment could even be improved compared to use case 3 as the relaxation of R_i allows *multiplexing* of m_{17} in slot 18.

(Table II, case 5): Maximizing slot and cycle availability
Similarly, slot and cycle availability can be optimized together in order to maximize the extensibility, i.e., more *future* messages having a high priority may be accommodated. Note, that m_{15} and m_{16} have been *multiplexed* in slot $S_{max} = 41$.

(Table II, case 6): Maximizing slot availability and slack
The combination of design metrics must not necessarily harmonize well as objectives can be contradictory. Here, λ_1 maximizes S_i (see use case 1) whereas λ_3 minimizes S_i and R_i (see use case 3), i.e., the result is a trade-off for both objectives. The slot optimization is worse than in use case 1, as there is no *multiplexing* available, the slack optimization performs worse than in use case 3 as only the repetition rate is minimized whereas the slot number is high because of λ_1.

TABLE II
RESULTS FOR DIFFERENT WEIGHTS λ_1, λ_2, λ_3 AND $B_i < R_i$.

Case	$(\lambda_1, \lambda_2, \lambda_3)$	Θ_{15}	Θ_{16}	Θ_{17}
1	$(1, 0, 0)$	$[41, 1, 2]$	$[40, B_{16}, R_{16}]$	$[41, 0, 2]$
2	$(0, 1, 0)$	$[S_{15}, B_{15}, 2]$	$[S_{16}, B_{16}, 4]$	$[S_{17}, B_{17}, 2]$
3	$(0, 0, 1)$	$[23, 0, 1]$	$[20, 0, 1]$	$[19, 0, 1]$
4	$(0, 0.5, 0.5)$	$[20, B_{15}, 2]$	$[19, B_{16}, 4]$	$[18, 1, 2]$
5	$(0.5, 0.5, 0)$	$[41, 1, 2]$	$[41, 2, 4]$	$[40, B_{17}, 2]$
6	$(0.5, 0, 0.5)$	$[41, 0, 1]$	$[40, 0, 1]$	$[39, 0, 1]$

In this section we discussed the fundamental effects of different weights on the schedule parameters $S_i, B_i, R_i \in \Theta_i$ using the proposed optimization procedure. Further, different weights can also be applied to individual messages $m_i \in M_{App}$ according to Eq. 7. Finally, we would like to note that our optimization framework can easily be extended by new user defined objectives that target other design requirements.

VI. CONCLUDING REMARKS

In this paper, we presented an incremental schedule synthesis approach for the DYN segment of FlexRay. The synthesized schedules satisfy the FlexRay protocol and meet specified real-time constraints. Our work considers the 64-cycle matrix properties of the FlexRay communication protocol while generating schedules for the DYN segment in an incremental manner. Further, we proposed an optimization procedure in order to retain schedules according to specific design metrics and illustrated its applicability by several design use cases. This work laid the groundwork towards schedule synthesis for the FlexRay DYN segment and it may be extended in multiple directions, e.g., schedule optimization towards more complex design objectives. Finally, we are interested in incorporating our scheduling framework into a tool-chain which may interact with industry accepted interfaces like Field Bus Exchange Format (FIBEX).

Acknowledgement: This work was supported in part by the DFG (Germany) through the SFB/TR28 Cognitive Automobiles.

REFERENCES

[1] AUTOSAR. Specification of FlexRay Interface, Ver. 3.0.3. www.autosar.org.
[2] BMW brake system relies on FlexRay. http://www.automotivedesignline.com/news/218501196, July 2009.
[3] Elektrobit Tresos. www.elektrobit.com.
[4] The FlexRay Communications System Specifications, Ver. 2.1. www.flexray.com.
[5] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *DAC*, 2007.
[6] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. FlexRay Schedule Optimization of the Static Segment. In *CODES+ISSS*, 2009.
[7] P. Pop, P. Eles, T. Pop, and Z. Peng. An approach to incremental design of distributed embedded systems. In *DAC*, 2001.
[8] T. Pop, P. Pop, P. Eles, and Z. Peng. Bus access optimisation for flexray-based distributed embedded systems. In *DATE*, 2007.
[9] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39:205–235, 2008.
[10] E.G. Schmidt and K. Schmidt. Message scheduling for the flexray protocol: The dynamic segment. *IEEE Trans. Vehicular Technology*, 58(5), 2009.
[11] S. Thiel and A. Hein. Modelling and using product line variability in automotive systems. *IEEE Software*, 2002.
[12] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *Int'l Conference on Application of Concurrency to System Design*, 2005.