Timing Modeling and Analysis for AUTOSAR OS Schedule Tables

Rongkun Yan*, Jian Guo*✉, Xiaoran Zhu†, Xin Li‡, Jizheng Ding*
*Soft/Hardware Co-design Engineering Research Center, East China Normal University
†National Trusted Embedded Software Engineering Technology Research Center
‡Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
Email: 51164500085@stu.ecnu.edu.cn, jguo@sei.ecnu.edu.cn

***Abstract*—Schedule table mechanism is an important character of AUTOSAR OS in addressing its real-time property and flexibility by providing an encapsulation of a static configuration with one or more actions, i.e., events set or tasks activation. As a real-time system, it is non-trivial to analyze schedulability of tasks in AUTOSAR. But it is intractable to analyze schedulability when many schedule tables run at the same time. Moreover, due to the complexity of behavior of schedule tables, the existing schedulability analysis cannot be applied to this mechanism. In this paper, we propose a model for AUTOSAR schedule table and give an effective algorithm to assist developers in analyzing schedule tables. A schedule table is formally modeled by a transition system. On the basis of the transition systems, we propose an algorithm to analyze schedulability by checking all the possible schedule scenarios. Furthermore, checking schedulability tool is implemented. Testbenches of schedule tables are executed both in our tool and an industrial AUTOSAR OS on microcontroller respectively. The result demonstrates our method is effective on analyzing schedulability of AUTOSAR OS.**

I. Introduction

Formal methods have been widely and successfully applied to functional properties verification. But unfortunately, it is difficult to verify some non-functional characteristics, which are the key to maintain validity in highly safety-critical real-time systems. Formal analysis on schedulability of tasks is researched. Liu et. al [1] first build a task model for periodic tasks with a period and execution time, then they discuss two scheduling algorithms over the model. Stigge et al. [2] describe a Digraph Real-Time (DRT) model to represent a task and study the feasibility problem on preemptive periodic tasks in a uniprocessor. For analyzing the temporal validity of real-time data objects, Wang et al. [3] give transaction model for tasks with three parameters, a fixed priority, worst-case execution time (WCET) and a relative deadline. The temporal validity of real-time data object is analyzed by periodic update transactions. S.Baruah [4] analyzes schedulability in a general 3-parameter sporadic task model [5] for mixed-criticality concurrent real-time tasks. Moreover, S.Baruah et al. [6] propose a scheduling model for periodic preemptive tasks inspired by control theory. C.Deutschbein et al. [7] present the problem of constructing cyclic executive upon multiprocessors.

AUTOSAR (AUTomotive Open System ARchitecture) [8] is an open and standardized layered software architecture and interfaces. The integration of software modules from different vendors forms the complete architecture. Researchers working on AUTOSAR OS are rich in progressive and proactive. There have been many studies on analyzing AUTOSAR OS specification [9]. Peng et al. adopt timed CSP method to model OS and the engine management system (EMS), some safety properties based on CSP models are verified through PAT in [10]. Huang et al. [11] apply formal semantics to describing tasks in AUTOSAR OS with time protection mechanism. The model predicts whether a task would violate its time constraint, and is implemented in Mathematica tool with a case study. Zhu et al. [12] focus on the timing properties of AUTOSAR OS and propose an automatic verification framework based on rewriting logic to analyze the timing behaviors. Therefore, many researches of AUTOSAR OS focus on its specification, implementation and verification. For schedulability of the AUTOSAR OS, both Zhao et al. [13] and Hatvani et al. [14] adopt preemptive threshold to improve the schedulability. Tasks are set with a higher priority and reduced execution time by the stack usage. In order to assist developers in assigning the priorities of tasks, Yoon et al. [15] present a real-time task chain model, which helps to obtain a near-optimal priority assignment from the model. They concentrate on the scheduling of tasks, but few researches deal with schedule tables. Wang et al. propose a method for generating schedule tables containing periodic tasks with dependence in [16] for the multi-core architectures. However the method does not give the correctness proof of generating schedule tables.

Among those aforementioned researchers, most of them analyze schedulability by modeling tasks, few attentions were given to schedule table mechanism [8], which is an important mechanism in AUTOSAR OS. A schedule table predefines a pattern of tasks activation by encapsulating a statically defined set of expiry points. The schedulability of schedule tables only depends on the experience of developers and is intractable to be analyzed manually. In order to address the schedulability of schedule tables, it requires a more specific model.

In this paper, we take into consideration the AUTOSAR OS scheduling with preemptive tasks and schedule tables in uniprocessor. We provide a formal model for abstracting behaviors of schedule tables and an algorithm to analyze schedulability by verifying all schedule scenarios. In addition, after analyzing the schedulability of schedule tables, we run testbenches of schedule tables on an implemented AUTOSAR OS based on TC1782 32-Bit Single-Chip Micro-controller [17]

to establish the effectiveness of our method.

The rest of this paper is organized as follows. Section 2 introduces AUTOSAR schedule tables and schedulability analysis. Section 3 describes formal models of schedule tables. Section 4 presents some definitions and notations. In Section 5, we propose a method by integrating concepts in section 4. Then, experimental result and analysis is given in Section 6. Finally, in Section 7, we conclude our work and describe future work.

II. Background

In this section, AUTOSAR schedule table and its scheduling algorithm are described. We also discuss schedulability problem for AUTOSAR schedule table.

A. Tasks, Alarms and Scheduling of AUTOSAR

AUTOSAR considers a task as the minimum unit to be scheduled. A simple application consists of a set of tasks. A task provides the framework for the execution of functions [18]. AUTOSAR provides two types of tasks: extended and basic tasks. Both types are components of complex control software and consist of code blocks. The main difference is that AUTOSAR OS receives and records request activations of a basic task already activated.

AUTOSAR OS offers at least one counter. Counters are represented by a counter value, measured in ticks [18]. Based on counters, AUTOSAR drives alarms and schedule tables. Each alarm has a predefined counter value and an action. An alarm will expire when the predefined value is reached. Then, the predefined actions will be executed, which may activate a task or set an event or call an alarm-callback routine. AUTOSAR offers two types of alarms: single and cyclic. Cyclic alarms satisfy the request of processing periodic task.

Each task in AUTOSAR assigns a priority and are scheduled according to their priority. AUTOSAR uses the static priority and first-come-first-served (FCFS) as its scheduling algorithm. The scheduler always choose the task with the highest priority to execute. If more than one tasks have the same priority, they are started depending on their order of activation. The priorities are distributed to tasks statically in AUTOSAR. AUTOSAR provides three difference scheduling policy: full preemptive, non preemptive and mixed of preemptive and non preemptive scheduling. The first means that a task in $running$ state will be transferred into $ready$ state when a higher priority task gets ready. The second means that once a task starts, it only release the processor when it terminate.

B. Schedule Table Mechanism of AUTOSAR

The schedule table mechanism is a new concept introduced in AUTOSAR. Schedule tables could statically define the pattern of tasks activation by encapsulating a predefined set of expiry points. An expiry point contains one or more actions, which can be either an activation of a task or a setting of an event. Each expiry point has a unique offset in ticks from the start of the schedule table. For a schedule table, the expiry points are processed in order of increasing offset. In addition, a schedule table has a specific duration which defines the modulus of the schedule table.

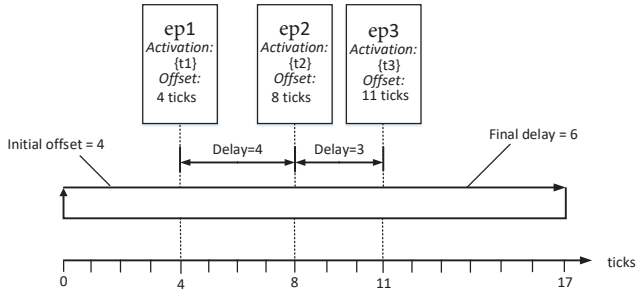


Fig. 1: An example repeating schedule table containing three expiry points, and t_1, t_2, t_3 is tasks.

We give an example in Figure 1 to illustrate the construction of schedule table. We denote this schedule table as st_1. This schedule table has the duration of 17 ticks. st_1 contains three expiry points ep_1, ep_2 and ep_3, where the initial expiry point ep_1 locates at 4 ticks from the start of st_1 with the action of activation of task t_1. ep_2 defines 8 as its offset and activation of task t_2 as its action. ep_3 defines 11 ticks as its offset and activation of task t_3 as its action. The delays between ep_1 and ep_2 is 4 ticks. The delay between ep_2 and ep_3 is 3 ticks. After ep_3, 6 ticks is delayed in schedule table st_1.

AUTOSAR offers two types of behavior of schedule tables: single-shot and repeating. On one hand, if a schedule table is configured as single-shot, it will stop final delay ticks after the final expiry point is processed. On the other hand, for a repeating schedule table, after the final expiry point has been processed, it loops back to the first expiry point. A repeating schedule table means that tasks is repeated at a period which equals to the schedule table duration.

C. Schedulability Analysis

The researches about schedulability analysis focus on the problem whether real-time tasks could satisfy its time property. We declare the meaning of schedulability: schedulability declare whether a set of schedule tables is schedulable. Schedulable means that all tasks activated in schedule tables could complete it execution before its deadline. Otherwise, schedule tables is non-schedulable.

In order to figure out the schedulability of real-time systems, several theories have been presented. We can categorize those methods by the existing scheduling algorithm. One of the most common algorithm is the earliest deadline first algorithm (EDF). EDF is considered as an optimized algorithm on preemptive uniprocessor to analyze a set of independent tasks characterized by activated time, deadline and worst case execution time. For such a dynamic priority scheduling algorithm, the schedulability analysis is co-NP in the strong sense [19].

Since our work focuses on the single processor platform with static priority scheduling, we pay attention to the schedulability analysis based on the response time analysis (RTA)

[20]. RTA is suited for fixed priority tasks system. This method calculate the worst case response time, then verify whether the worst case response time of a task is smaller than its deadline.

III. Modeling of schedule tables

In order to abstract behaviors of schedule tables, we need a deterministic specification of AUTOSAR firstly. But AUTOSAR does not provides a single specification, but multiple features to satisfy various requirements. For instance, some implemented AUTOSAR OS only accept activated request from an unactivated task, while others could record activated requests of an activated task. Among those features, our model only aims at the following characteristics:

- Tasks are allowed to share the same priority level.
- We take no account of extended tasks and events.
- Scheduling policy is full preemptive scheduling.
- A task can be activated infinitely many times.

For convenience of expression, if a task is assigned to more than one expiry point, we assume it as different tasks. In additional, we assume all schedule tables are repeating. In fact, a repeating schedule table could be transferred to a single-shot schedule table by converting the final delay into infinitely great. This section proposes a formal models for schedule tables, and an example is given.

A. Digraph Schedule Table Models

We use a directed digraph to represent a schedule table and named it Digraph Schedule Table model (DST). A DST model abstracts the attributes of a schedule table, e.g., the set of tasks, the set of expiry points and the order expiry points to be processed.

Definition 1. *Each DST is a 5-tuple* $(Tasks,\ EPs,\ Act,\ Next,\ Delay)$*, where*

- $Tasks$ *is a finite set of tasks. Each task is described as a 3-tuple* (e, d, l)*. Assuming* $t_k \in Tasks$*, then* $e(t_k), d(t_k), l(t_k)$ *denote the execution time, deadline and priority level of* t_k *respectively (lower number for higher priority).*
- EPs *is a finite set of expiry points.*
- $Act : EPs \rightarrow 2^{Tasks}$ *is a mapping function. This function shows that an expiry point contain a set of tasks to activate.*
- $Next : EPs \rightarrow EPs$ *is a transition relation to describe the order expiry points to be processed. Due to the linear execution of schedule table, each expiry point has one and only one post expiry point.*
- $Delay : EPs \rightarrow ticks$ *is a function to describe the difference between the offset of two adjacent expiry points.* $ticks$ *is represented by a positive number. We calculate* $Delay$ *of the last expiry point as the final delay plus the initial offset.*

Note that the initial offset is concealed in DST. Because during the process of repeating schedule tables running, the initial offset always can be considered as a part of the last expiry point delay, except the first loop. With regard to the definition of $Delay$, the duration of a schedule table $\delta(DST)$ can be obtained by

$$\delta(DST) = \sum_{ep' \in EPs} Delay(ep').$$

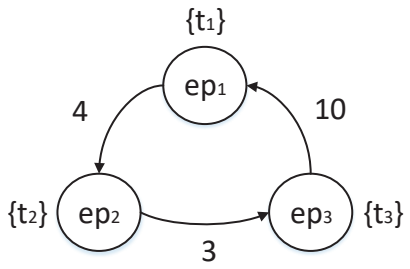


Fig. 2: Digraph Schedule Table model dst_1 for st_1, where $t_1 =< 2, 4, 2 >,\ t_2 =< 2, 3, 1 >\ \ t_3 =< 2, 9, 6 >$.

B. Execution of DSTs

When a DST starts to run, it does not ever stop, a fragment is named it a path ρ during the infinite running of a DST. For a path, the same expiry point might appear multiple times and we distinguish them by the expire time. We group an expiry point ep_i and the expire time r_i into a pair (ep_i, r_i) and named it an instance of an expiry point. Naturally, a path consists of an finite sequence of instances of expiry points, shown as, $\rho = [(ep_1, r_1), (ep_2, r_2), \ldots, (ep_k, r_k)]$.

When an expiry point ep_i arrives, tasks in $Act(ep_i)$ will be activated. When a task is activated, there is an instance of the task, called a job. A 4-tuple (e_i, d_i, l_i, r_i) denotes a job with execute time e_i, deadline d_i, priority l_i and activated time r_i. For a path, expire times are constrained by the delay of corresponding expiry points, i.e.,

$$r_{i+1} = r_i + Delay(ep_i).$$

Similar to calculate duration of schedule tables, we get the length of paths by calculating the difference between the first and the last expiry time. For a path $\rho = [(ep_1, r_1), (ep_2, r_2), \ldots, (ep_k, r_k)]$, the length is calculated as follows:

$$length(\rho) = r_k - r_1.$$

C. Example Model

We construct the model for the schedule table st_1 as a digraph schedule table model in Figure 2 and give the tuple dst_1 as below.

$dst_1 = \{Tasks_1, EPs_1, Act_1, Next_1, Delay_1\}$, where

- $Tasks_1 = \{t_1,\ t_2,\ t_3\}$ which contains all tasks the schedule table can activate.
- $EPs_1 = \{ep_1,\ ep_2,\ ep_3\}$ which contains expiry points.
- Act_1 refers to actions of an expiry point.$Act_1(ep_1) = \{t_1\}$, $Act_1(ep_2) = \{t_2\}$, $Act_1(ep_3) = \{t_3\}$.
- $Next_1$ shows the order that expiry points are processed. $Next_1(ep_1) = ep_2$, $Next_1(ep_2) = ep_3$, $Next_1(ep_3) = ep_1$.

- $Delay_1$ is defined by $Delay_1(ep_1) = 4$, $Delay_1(ep_2) = 3$, $Delay_1(ep_3) = 10$.

IV. Definition and notation

As preparations, some concepts are introduced in this section. Firstly, the schedulability is defined.

Definition 2. *A task is schedulable in an operating system if and only if the task always finish before its deadline in any scenario in a certain scheduling algorithm. The system is called schedulable if and only if every tasks in this system are schedulable.*

Under the schedulable definition, we now focus on checking whether a single task always satisfies its deadline. Each task is decomposed into many jobs. The main idea of our method is to check, for a job, whether the time interval between activation and deadline is big enough to accommodate its execution and other jobs which interrupt this job. A question is proposed, how to recognize jobs which could interrupt a specific job? In order to solve this question, The concept of the computational requirement is defined.

A. Computational Requirement Function

The AUTOSAR OS adopts the static priority as first scheduling algorithm and first come first served as second scheduling algorithm. So a job could not be interrupted by jobs with lower priorities, or by jobs with the same priority but are activated later. Based on this fact, for an instance of expiry point, the computational requirement function is defined to sum the execution time of jobs which could interrupt a specific job.

Definition 3. *For a job (e, d, l, r) and an instance of an expiry point (ep_i, r_i), the computational requirement function $\omega_{l,r}^{(ep_i,r_i)}$ sums the execution time of jobs which could interrupt the job (e, d, l, r).*

If $r_i > r$, jobs from (ep_i, r_i) are activated later. Then we only sum execution time of jobs which have a higher priority than l.

If $r_i \leq r$, we sum execution time of jobs which have priorities higher than or equal to l:

$$\omega_{l,r}^{(ep_i,r_i)} = \sum_{task \in \xi_{l,r}^{(ep_i,r_i)}} e(task), \tag{1}$$

where

$$\xi_{l,r}^{(ep_i,r_i)} = \begin{cases} \{t_i | t_i \in Act(ep_i) \wedge l(t_i) < l\} \\ \qquad\qquad if\ r_i > r \\ \{t_i | t_i \in Act(ep_i) \wedge l(t_i) \leq l\} \\ \qquad\qquad if\ r_i \leq r \end{cases}$$

The computational requirement function $\omega_{1,4}^{(ep_2,3)}$ is given as an example. $\omega_{1,4}^{(ep_2,3)}$ sums execution time of jobs which can interrupt the job $(e, d, 1, 4)$ in the instance of expiry point $(ep_2, 3)$. Because the instance of expiry point arrives earlier than the job, i.e., $3 < 4$, jobs which have priorities equal to or higher than 1 are considered. t_2 is the only one task satisfies the condition in ep_2, so $\omega_{1,4}^{(ep_2,3)}$ returns $e(t_2)$, i.e., 2 as its value.

The computational requirement function for a path is given in Definition 4.

Definition 4. *For a job (e, d, l, r) and a path $\rho = [(ep_1, r_1), (ep_2, r_2), \ldots, (ep_n, r_n)]$, the computational requirement function $\Omega_{l,r}(\rho)$ sums the execution time of jobs during the path ρ but excludes jobs which cannot interrupt the job (e, d, l, r):*

$$\Omega_{l,r}(\rho) = \sum_{i=1}^{n} \omega_{l,r}^{\rho_i}, \tag{2}$$

where ρ_i represents the ith instance of expiry point that emerges on the path ρ. For example, ρ_1 is (ep_1, r_1) and ρ_2 is (ep_2, r_2).

B. Request Function

The computational requirement function helps a path to calculate the execution time about a job. But an execution of a DST is infinite and so is the paths. Tasks in a schedule table appear infinite times in the path. But most of paths are meaningless. We define the request function to filter out meaningful paths.

1) Prefix Request Function: In order to get the meaningful paths, we add two limiting conditions on paths: 1) the first expiry point of paths, 2) the maximum length of paths. The prefix request function [21] is extended to calculate the computational requirement of paths about jobs under two limitation.

Definition 5. *For an expiry point ep_i in a DST, the prefix request function about a job (e, d, l, r) $prf_{l,r}^{ep_i}(\theta)$ calculates the maximum value of $\Omega_{l,r}(\rho)$ where ρ starts with $(ep_i, 0)$ and $length(\rho) < \theta$. The prefix request function is defined as below:*

$$\begin{aligned} prf_{l,r}^{ep_i}(\theta) = & max\{\Omega_{l,r}(\rho) | \\ & \rho\ start\ by\ (ep_i, 0)\ \wedge\ length(\rho) < \theta\}. \end{aligned} \tag{3}$$

The $prf_{l,r}^{ep_i}(\theta)$ describes a scenario that a job (e, d, l, r) is activated at the rth tick after an instance of ep_i. The result of $prf_{l,r}^{ep_i}(\theta)$ represents the total execution time of jobs that can interrupt the job (e, d, l, r) in this scenario. Since no path has length less than 0, we dictate θ must be greater than 0.

For the DST in Figure 2, an example of the prefix request function about a job $(e, d, 4, 0)$ $prf_{4,0}^{ep_1}(\theta)$ is shown in Figure 3. When $0 < \theta \leq 4$, there is only one path $[(ep_1, 0)]$ can satisfies the limitations. So the value of the prefix request function equals to computational requirement of the path as 2. When $4 < \theta \leq 7$, the ordinate value equal to the computational requirement of path $[(ep_1, 0), (ep_2, 4)]$ which is 4. When $7 < \theta \leq 17$, the path $[(ep_1, 0), (ep_2, 4), (ep_3, 7)]$ also satisfies the limitations. But the task $t_3$in ep_3 has a lower priority, so the prefix request function ignores it and keep return 4 until θ exceeds 17. As we can see, the prefix request function is a monotone increasing function of the length of paths.

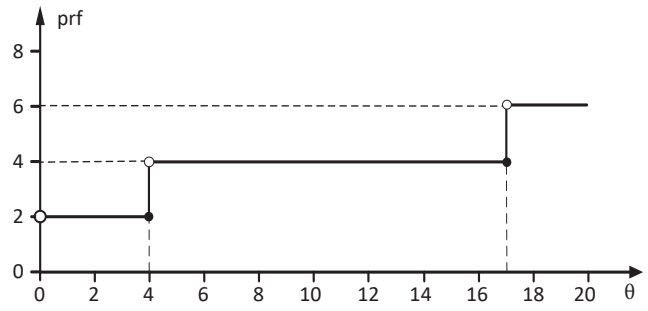


Fig. 3: An example for the prefix request function $prf_{4,0}^{ep_1}(\theta)$ according to dst_1 in Figure 2. The abscissa represents the maximum length of paths. The ordinate represents value of the prefix request function.

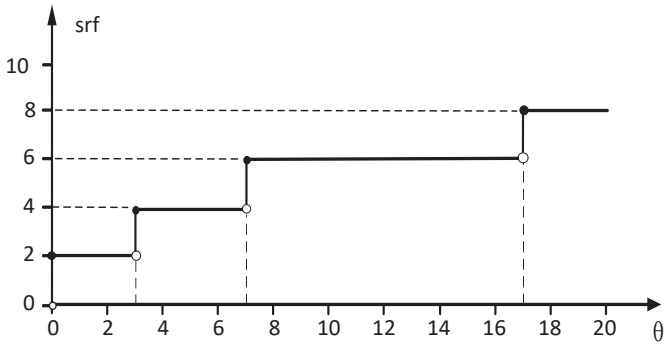


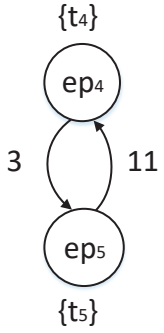
Fig. 4: An example for the suffix request function $srf_6^{ep_3}(\theta)$ according to dst_1 in Figure 2. The abscissa represents the maximum length of paths. The ordinate represents value of the suffix request function.

2) Suffix Request Function: Besides constraining the first expiry point of paths, we also take into account paths which end up with a specific expiry point. The suffix request function limits the last expiry point of paths.

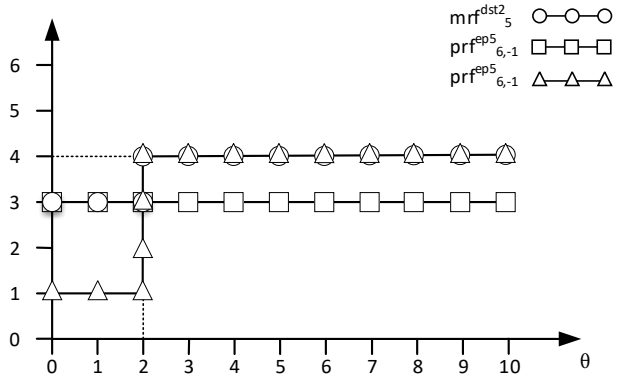
Definition 6. *We define the suffix request function $srf_l^{ep_i}(\theta)$ to calculate the maximum value of $\Omega_{l,r}(\rho)$ where ρ ends up in the instance of expiry point (ep_i, r) and $length(\rho) < \theta$. The suffix request function is defined as below:*

$$srf_l^{ep_i}(\theta) = max\{\Omega_{l+1,-1}(\rho) | \rho \ is \ ended \ in \ (ep_i, r_i) \ \wedge \ length(\rho) \leq \theta\}. \quad (4)$$

The suffix request function describes a scenario that a job (e, d, l, r) is activated together with an instance of expiry point (ep_i, r). Since (ep_i, r) is at the end of the path, jobs on the path are earlier that the job (e, d, l, r), so jobs which have priorities higher than or equal to l can interrupt the job (e, d, l, r). The suffix request function ignores the expire time and uses the $\Omega_{l+1,-1}(\rho)$ to sum the execution time of jobs with priorities higher than or equal to l.



(a) An example of dst_2 which contains two expiry points and $t_4 =< 1, 3, 3 >,\ t_5 =< 3, 8, 5 >$.



(b) Maximum request function $mrf_5^{dst_2}(\theta)$. The abscissa represents the maximum length of paths. The ordinate represents value of the prefix request function and the maximum request function.

Fig. 5: An example of demonstrating the relation between the prefix request function and the maximum request function.

For an example, in the DST in Figure 2, the suffix request function about a job $(e, d, 6, r)$ $srf_6^{ep_3}(\theta)$ is given in Figure 4. When $0 \leq \theta < 3$, there is only one path $[(ep_3, r_3)]$ satisfies the limitations. The value of suffix request function equals to computation requirement of the path as 2. When $3 \leq \theta < 7$, the ordinate value equal to the computational requirement of path $[(ep_2, r_2), (ep_3, r_3)]$ as 4. When $7 \leq \theta < 17$, $[(ep_1, r_1), (ep_2, r_2), (ep_3, r_3)]$ satisfies the limitations and makes the maximum execution time 6 as the value of suffix request function.

3) Maximum Request Function: Now we try to relax the limitations of paths and only limit the the maximum length of paths. After lifting restriction on expiry points, the maximum request function is given as below.

Definition 7. *For a digraph schedule table model dst_i, the maximum request function $mrf_l^{dst_i}(\theta)$ calculates the maximum value of $prf_{l+1,-1}^{ep_i}(\theta)$ where $ep_i \in EPs_i$. The maximum request function is defined as below:*

$$mrf_l^{dst_i}(\theta) = max\{prf_{l+1,-1}^{ep_i}(\theta) | ep_i \in EPs_i\}. \quad (5)$$

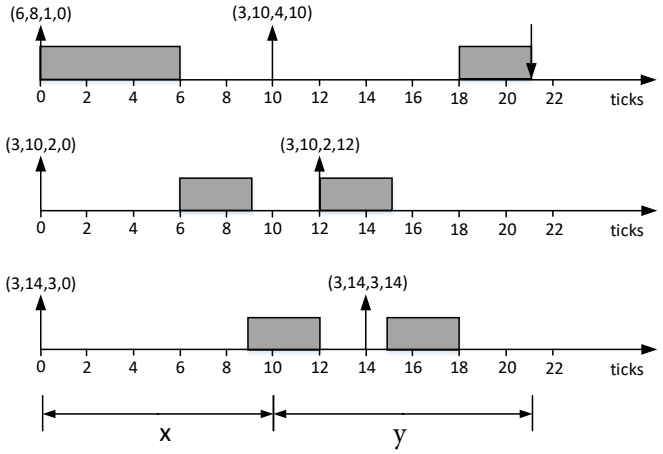


Fig. 6: Example demonstrates a schedule scenario which makes a job $(3, 10, 4, 10)$ misses its deadline.

Where we use $prf_{l+1,-1}^{ep_i}(\theta)$ to sum the execution time of jobs with priorities higher than or equal to l.

An example of a maximum request function $mrf_5^{dst_2}(\theta)$ is shown in Figure 5b. A new digraph schedule table model dst_2 is given in Figure 5a. The dst_2 contains two expiry points, so there are two prefix request functions $prf_{6,-1}^{ep_4}$ and $prf_{6,-1}^{ep_5}$ in Figure 5b. When $0 < \theta \leq 2$, the value of $prf_{6,-1}^{ep_5}$ is bigger. But when $2 < \theta \leq 10$, the value of $prf_{6,-1}^{ep_4}$ is bigger and offers the value of the maximum request function.

C. Busy Window

Tasks in a real-time operating system must finish before their deadline, or a time fault will occur. But identifying the task which causes time fault may be far from obvious. See Figure 6 for an example of this. For the job $(3, 10, 4, 10)$, the time interval between its activation and deadline is big enough to accommodate its execution and tasks in the other paths, but it cannot finish before its deadline. This is because the job $(6, 8, 1, 0)$ takes too long to execute and blocks the execution of $(3, 10, 2, 0)$., then the job $(3, 14, 3, 0)$ is blocked too. The influence spreads throughout the running of jobs, until the job $(3, 10, 4, 10)$ finishes in 21, which means it misses its deadline.

So, it is not sufficient to only focus on the time interval after the activation of $(3, 10, 4, 10)$. We extend the time interval towards to the past. But how far should we extend? In order to find a solution to this problem, we introduce the well-known concept: busy window extension technique [22].

Busy window for priority level l is a time interval where the processor continuously runs jobs which have priority l or higher priorities. If two jobs are not in the same busy window, the processor must have executed jobs with lower priority or been idle some time between them. A job (e, d, l, r) can only be affected by jobs within the busy window. If finding the maximum busy window for l, then we can determine how many ticks should we add into the time interval. The maximum busy window for l is represented by σ_l.

Lemma 1. *For a DST set $\Gamma = \{dst_1, dst_2, \ldots, dst_n\}$, the maximum busy windows for l can be calculated by finding the smallest positive number θ which satisfy*

$$\sum_{dst_i \in \Gamma} mrf_l^{dst_i}(\theta) = \theta. \tag{6}$$

Proof. According to the definition of busy window, each DST has a path starts together with the maximum busy windows. So we need find the first expiry point of paths that can makes the maximum execution time.

If there is a θ satisfies Equation 6, then for each maximum request function, at least one prefix request function offer their value at θ. We pick any one prefix request function from a maximum request function, then a set of expiry points is obtained as superscript of those prefix request functions. When those expiry points expire at the same time, we can get the maximum busy windows for l.

Now assuming we use this method and get an expiry point ep' in dst' as the first expiry point of a path. But there is another expiry point ep'' in dst' which could make a longer busy window. Because the prefix request function of ep'' does not offer the value of $mrf_l^{dst'}$ at θ, i.e.,

$$prf_{l+1,-1}^{ep''}(\theta) < prf_{l+1,-1}^{ep'}(\theta).$$

When we choose ep'' instead of ep' to expire with other expiry points simultaneously, the sum of the prefix request function is less than θ, i.e.,

$$prf_{l+1,-1}^{ep''}(\theta) + E < prf_{l+1,-1}^{ep'}(\theta) + E = \theta, \tag{7}$$

where $E = \sum_{dst \in \Gamma - dst_1} mrf_l^{dst}(\theta)$.

When ep'' expires with other expiry points, the busy window is shorter than θ. Therefore,selecting ep'' can lead to a contradiction. There does not exist this expiry like ep''. □

In some non-schedulable cases, the maximum busy windows is infinitely great. In order to finish computation, the hyperperiod of a set of DSTs is defined as follows.

Definition 8. *For a set of DSTs $\Gamma = \{dst_1, dst_2, \ldots, dst_n\}$, let durations of DSTs be $\delta_1, \delta_2, \ldots, \delta_n$ respectively. The hyperperiod of τ is defined as the least common multiple of all durations, i.e., $lcm(\delta_1, \delta_2, \ldots, \delta_n)$.*

Lemma 2. *A finite set of DSTs is non-schedulable, if there exists a busy window exceeds the hyperperiod of them.*

Proof. The infinite running of a set of DSTs has to be a cyclic and the period is the hyperperiod of them. If any busy window exceeds the hyperperiod, the jobs which are activated in the hyperperiod cannot finish in current hyperperiod. In the next hyperperiod, except the jobs are activated in the new hyperperiod, the unfinished jobs also need to execute. So the tasks with the lowest priority must be blocked. By parity of reasoning, the task with the lowest priority must violate its deadline at some point. □

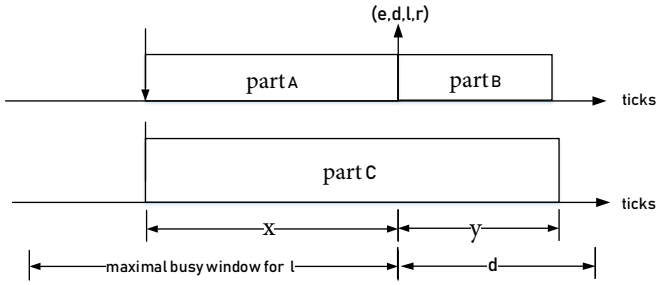


Fig. 7: An schedule scenario for analyzing job (e, d, l, r).

D. Schedulability Analysis of A Single Task

After definition of conceptions, we come back to the concerned question, i.e., how to check a job always meeting its deadline. As mentioned in Section IV-C, the main analysis idea about a job is to extend the time interval between its activation and deadline by adding x ticks into the past. For each $x \in (0, \sigma_l]$, we check the extended time interval is big enough to accommodate its execution and other jobs which interrupt this job.

Notice that different paths give different interference jobs experience. For instance, if the first path in Figure 6 starts with the job $(3, 10, 4, 0)$, i.e., shifts 10 ticks to the left, then the job might meet its deadline. We select one expiry point in each DST to determine a certain interference to a job. For an example, the interference in Figure 6 can be determined by $\{(6, 8, 1), (3, 10, 2), (3, 14, 3)\}$. The combination of expiry points is defined as below.

Definition 9. *For a set of DSTs $\Gamma = \{dst_1, dst_2, \ldots, dst_n\}$, the combination of expiry points $\Phi(\Gamma)$ represents the Cartesian product between EPs of every DSTs. We define the combination of expiry points as:*

$$\Phi(\Gamma) = EPs_1 \times EPs_2 \times \cdots \times EPs_n. \tag{8}$$

The interference of a job within the extended time interval may be divided into three parts: 1) jobs before (e, d, l, r) from the same path, 2) jobs after (e, d, l, r) from the same path, 3) jobs from other paths. The three parts correspond to the part A, part B and part C in Figure 7.

Then the workload of three parts is calculated respectively. If we only concern the path in part A, the path just ends up in job (e, d, l, r). In this way, suffix request function with $\theta = x$ provides the most perfect way to deal with part A. For part B, a similar pattern is employed. Since the path in part B starts with the job (e, d, l, r), the prefix request function with $\theta = y$ is calculated part B. For part C, the combination of expiry points is calculated. Because each element in combination of expiry points represents a set of first expiry point of paths in part C. The workload of part C is computed by the prefix request function with $\theta = x + y$. Notice that the path in part B of the prefix request function starts with (ep_i, x) instead of $(ep_i, 0)$.

As a result, other jobs may interrupt this job (e, d, l, r) during calculating workload of three parts. Among the three parts, the job (e, d, l, r) is the last one to finish because of its lowest priority. If the time interval with length $x + y$ could accommodate the total workload when $y \in (e, d]$, then the job meets deadline. Since a job is an instance of a task, we generalize checking jobs for tasks to consider all activated time for the task in the expiry point ep_i.

Theorem 1. *For a finite set of DSTs Γ, if there exists a task t_i activated for the digraph schedule table model dst_i. Then task t_i always meets its deadline if:*

$$\begin{aligned}\forall \varepsilon \in \Phi(\Gamma - \{dst_i\}), x \in [0, \sigma_{l(t_i)}] : \exists y \in (e(t_i), d(t_i)] : \\ srf^{ep_i}_{l(t_i)}(x) + prf^{ep_i}_{l(t_i),-1}(y) - \omega^{(ep_i,x)}_{l(t_i),-1} \\ + \sum_{ep' \in \varepsilon} prf^{ep'}_{l(t_i),x}(x + y) \leq x + y.\end{aligned} \tag{9}$$

Proof. The formula $\omega^{(ep_i,x)}_{l(t_i),-1}$ is the overlap between $srf^{ep_i}_{l(t_i)}$ and $prf^{ep_i}_{l(t_i),-1}$. Assuming Equation 9 evaluates to false, but task t_i always meets its deadline. Since Equation 9 is false, there must exist $\varepsilon \in \Phi(\Gamma - \{dst_i\})$ and $x \leq \sigma_{l(t_i)}$ to make the value of Equation 10 larger than $x + y$, where $y = d(t_i) + 1, d(t_i) + 2, \ldots, d(t_i)$.

$$\underbrace{srf^{ep_i}_{l(t_i)}(x)}_{partA} + \underbrace{prf^{ep_i}_{l(t_i),-1}(y)}_{partB} - \underbrace{\omega^{(ep_i,x)}_{l(t_i),-1}}_{overlap} + \underbrace{\sum_{ep' \in \varepsilon} prf^{ep'}_{l(t_i),x}(x + y)}_{partC}. \tag{10}$$

Equation 10 accumulates the execution time of task t_i and tasks affecting. That is, t_i is the last one to be finished among them. Therefore, the minimal y makes Equation 10 $\leq x + y$ to be the worst-case response time (WCRT). But due to the inexistence of $y \in (e(t_i), d(t_i)$ must satisfy Equation 9. The WCRT exceeds deadline. When expiry points in ε expire at 0 and t_i is activated at x, task t_i does not meet its deadline, leading to a contradiction.

Assuming Equation 9 holds, but a job j_i of task t_i violates its deadline. There exists a group of jobs could interrupt j_i. In this group, the first activated job is activated x ticks earlier than j_i. According to Lemma 1, $x \leq \sigma_{l(t_i)}$. Since the Equation 9 holds, the workload of this group is less than or equals to $x + y$. Therefore the response time of j_i is less than or equals to y, leading to a contradiction. □

V. Schedulability Analysis Method

Some conceptions for analyzing schedulability of a single task are defined in Section IV. We propose an integrated method based on these conceptions.

A. Steps of Method

For a set of DSTs $\Gamma = \{dst_1, dst_2, \ldots dst_n\}$, the process of analyzing schedulability of those digraph schedule table models is divided into eight stages:

1) Picking up a task with the lowest priority from $\tau = Task_1 \cup Task_2 \cup \cdots \cup Task_n$ and denoting it as t_i.
2) dividing Γ into two parts: i) the one is that a DST includes the task t_i, ii) the set of other DSTs. They are denoted by $ldst$ and $\Gamma - ldst$ respectively.
3) Calculating the maximum busy windows for $l(t_i)$ denoted by $\sigma_{l(t_i)}$. The set of DSTs is non-schedulable if $\sigma_{l(t_i)}$ exceed hyperperiod, .
4) Picking one element ε from $\Phi(\Gamma - ldst)$, assuming the expiry points in ε are the first expiry point of paths.
5) The task t_i is schedulable if the equation 11 is satisfied, where the task $t_i \in Act(ep_i)$.

$$\forall x \in [0, \sigma_l t_i] : \exists y \in (e(t_i), d(t_i)] : srf^{ep_i}_{l(t_i)}(x) + prf^{ep_i}_{l(t_i),-1}(y) - \omega^{(ep_i,x)}_{l(t_i),-1} \sum_{ep' \in \varepsilon} prf^{ep'}_{l(t_i),x}(x+y) \leq x + y \tag{11}$$

6) If the task t_i is schedulable, then we move to the stage (4) with other $\varepsilon \in \Phi(\Gamma - ldst)$. Otherwise, if there exist a scenario makes Equation 11 false, we declare this set of DSTs is non-schedulable. If every elements in $\Phi(other\_st)$ have been verified, we move to the next step.
7) Let $\tau = \tau - t_i$, picking one task with the lowest priority in this set except t_i. Then repeating the procedure from step 2 until τ is empty.
8) If all tasks pass the validation, declaring this set of DSTs is schedulable.

B. Integrated Method

Two primary algorithms are employed to implement the method as below. There are two global variables: 1) Γ represents the set of DSTs, 2) τ represents the set of all tasks in Γ.

Algorithm 1 SCHEDULABILITY

Input:
Output: schedulable or non-schedulable
1: **if** $\tau = \varnothing$ **then**
2: **return** schedulable
3: **else**
4: $t_i \leftarrow$ lowest priority task in τ
5: **if** $pass = TASK\_ANALYSIS(t_i)$ **then**
6: $\tau \leftarrow \tau - t_i$
7: **return** SCHEDULABILITY()
8: **else**
9: **return** non-schedulable
10: **end if**
11: **end if**

The algorithm 1 is finding an non-schedulable task, or proving the set of DSTs is scheduled. The line 1, 2 check whether the algorithm 1 should terminate. The line 4 implement the stage 2 which aims to select the task with the lowest priority. The function $TASK\_ANALYSIS$ verifies whether a task meets its deadline. If this function returns $pass$, then the task is removed from τ and another task is checked in the algorithm. τ is empty means all tasks are schedulable. Therefore the set of DSTs is schedulable too. If the return of $TASK\_ANALYSIS$ is $not - pass$, then the task is not schedulable and Γ is also non-schedulable. The detail of $TASK\_ANALYSIS$ is shown in Algorithm 2.

Fig. 8: An example of DST dst_3, where $t_6 =< 2, 11, 6 >$, $t_7 =< 1, 3, 4 >$.

Algorithm 2 TASK_ANALYSIS

Input: t_i
Output: pass or not-pass
1: **if** $\sigma_{l(t_i)} \leftarrow BOUNDARY(l(t_i)) > \prod_{dst' \in \Gamma} \delta(dst')$ **then**
2: **return** not-pass
3: **end if**
4: **for** $\forall \quad \varepsilon \in \Phi(\Gamma - ldst)$ **do**
5: **for** $\forall \quad x \in [0, \sigma_{l(t_i)}]$ **do**
6: **if** $!\exists y \in (e(t_i), d(t_i)) : srf^{ep_i}_{l(t_i)}(x) + prf^{ep_i}_{l(t_i),-1}(y) - \omega^{(ep_i,x)}_{l(t_i),-1} + \sum_{ep' \in \varepsilon} prf^{ep'}_{l(t_i),x}(x+y) > x + y$ **then**
7: **return** not-pass
8: **end if**
9: **end for**
10: **end for**
11: **return** pass

Algorithm 2 aims to verify whether task t_i meets its deadline. ep_i represents the expiry point that the task t_i is included. In line 1, the maximum busy windows is computed and compared with the hyperperiod. Line 4 to 7 is an implement action of Equation 9. srf and prf represent the suffix request function and the prefix request function respectively. If the algorithm finds at least one scenario makes t_i to miss it deadline, then returns $not - pass$.

C. Demonstration

We give an example to illustrate how to decide the schedulability of a set of schedule tables. There is a set of DSTs by uniting dst_1 in Figure 2, dst_2 in Figure 5a and a new DST dst_3 shown in Figure 8. In this case, the set of DSTs $\Gamma = \{dst_1, dst_2, dst_3\}$. The set of tasks τ is:

$$\tau = Tasks_1 \cup Tasks_2 \cup Tasks_3 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}.$$

Picking a task with the lowest priority in τ, denoting as t_i. We choose the lowest priority task t_3. Task t_3 divides Γ into two parts: dst_1 and $\{dst_2, dst_3\}$, denoted by $ldst$ and $\Gamma - ldst$ respectively. Then, the maximum busy windows for $l(t_3)$ is calculated by Equation 6 and denoted it as σ_6 = 13. In the next step, $\Phi(\Gamma)$ is all combinations of expiry points in $\Gamma - ldst$ as $\{\{ep_4, ep_6\}, \{ep_5, ep_6\}\}$. We pick $\varepsilon = \{ep_4, ep_6\}$ firstly. Then we structure a scenario with two paths: $\rho_1 = [(ep_4, 0), (ep_5, 3), (ep_4, 14) \dots]$ for dst_2 and $\rho_2 = [(ep_6, 0), (ep_6, 20), (ep_6, 40), \dots]$ for dst_3. Then we verify whether there always exists a $y \in (0, d(t_3)]$ for any $x \in [0, \sigma_6]$ to satisfy Equation 11, i.e.,

$$\forall x \in [0, 13] : \exists y \in [2, 9] : \\ sr f_6^{ep_3}(x) + pr f_{6,-1}^{ep_3}(y) - \omega_{6,-1}^{(ep_3,x)} + \sum_{ep_i \in \varepsilon} pr f_{6,x}^{ep_i}(x + y) \leq x + y.$$

When x = 0, there is y = 9 to satisfy the equation, when x = 1, there is y =8 to satisfy the equation, ..., x = 13, there is y =3 to satisfy the equation. So the task t_3 meets its deadline in this scenario and is deleted from τ. Then the task t_6 is picked as the lowest priority task in τ and repeat those steps. Finally, the algorithm stops when τ is empty. We conclude that DSTs is schedulable.

VI. Experimental analysis

A schedulability tool is implemented based on the method. The tool accepts a set of digraph schedule table models as input and returns "schedulable" or "non-schedulable" as the result.

In order to validate the effectiveness of the schedulability tool, an experiment is set to compare the result of method and the execution of an implemented AUTOSAR OS in a board. In this experiment, a program produces a set of schedule tables and implement on the board. In this comparison an industrial OS ofAUTOSAR is employed on the hardware platform TC1782 32-Bit Single-Chip Microcontroller [17]. First the tool implemented the schedulability algorithm verifies the set of schedule tables whether to satisfy the schedulability. Then, this set of schedule tables execute on the board. In the end. Comparison of results of the tool and board is analyzed.

A. Experiment Setup

Schedule tables are generated subject to seven parameters as below. Table I shows the ranges of parameters.

- st_{num}, the number of schedule tables.
- ep_{num}, the maximum number of expiry points in a schedule table.
- $task_{num}$, the maximum number of tasks in an expiry point.
- $Delay$, the delay of an expiry point.
- $e(t)$, the execution time of a task.
- $d(t)$, the deadline of a task.
- δ, the duration of schedule tables.

After generating schedule tables, they are transformed into DSTs and are considered as the input of the schedulable tool. Meanwhile those schedule tables are deployed into an implemented AUTOSAR by a configuration tool and test the schedulability.

st_{num}	ep_{num}	$task_{num}$	$Delay$	$e(t)/d(t)$	$d(t)/\delta$
$[2, 6]$	$[1, 4]$	$[1, 3]$	$[10, 30]$	$[0, 0.16]$	$[0.1, 0.5]$

TABLE I: Ranges of parameters

B. Consistency Analysis

The experiment of comparison concludes that the result of the schedulable tool is consisted exactly with an actual running on the implemented AUTOSAR OS in the hardware platform. Several examples and their result are synthesized in Table II. $Task_{sum}$ represents the amount of tasks.

		Schedulability	
st_{num}	$task_{sum}$	schedulable tool	industrial OS
3	14	Invalid	Invalid
4	28	Valid	Valid
5	36	Invalid	Invalid
6	27	Invalid	Invalid

TABLE II: Comparison of schedulability analysis result

If the schedulability in the schedulable tool is non-schedulable, an amending advice will be given. For example, when we test task t_7, Equation 9 could not be satisfied when $x = 0$ and $\varepsilon = ep_4, ep_6$. Then we try to widen its deadline from 3 ticks to 4 and test t_7 again, until the task passes validation. Finally, we obtain an amended set of schedule tables, while it is schedulable in the tool and also on implemented AUTOSAR OS.

VII. Conclusion

In this paper, schedulability of schedule tables in AUTOSAR OS is addressed, A directed graph, called DST, models a schedule table and abstract the behaviors of schedule tables as formal models. An algorithm is proposed based on DST models. The algorithm analyzes the schedulability of schedule tables by covering all the possible schedule scenarios. A tool about the algorithm is implemented. Moreover, we conduct an experiment, which runs a set of schedule tables on an industrial AUTOSAR OS and the tool respectively to check the effectiveness of the method. Through the comparison result, we believe that the method could perform well in analyzing schedule tables.

This work has gotten a promising result in analyzing schedule tables on uniprocessor. Since AUTOSAR specification has introduced multicore operating systems, analyzing schedulability of schedule tables on multiprocessor are important. Basic tasks are taken into consideration in the paper, there are two types tasks: basic and extended task. Schedulability analysis for extended tasks, which are allowed to synchronize through setting and waiting events, is another interesting topic we would like to pursue.

REFERENCES

[1] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[2] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 71–80, 2011.

[3] Jiantao Wang, Kam-yiu Lam, Song Han, Sang Hyuk Son, and Aloysius K. Mok. An effective fixed priority co-scheduling algorithm for periodic update and application transactions. *Computing*, 95(10-11):993–1018, 2013.

[4] Sanjoy Baruah. Schedulability analysis for a general model of mixed-criticality recurrent real-time tasks. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*, pages 25–34, 2016.

[5] Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 2017.

[6] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Victor Verdugo. A scheduling model inspired by control theory. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, pages 78–87, 2017.

[7] Calvin Deutschbein, Tom Fleming, Alan Burns, and Sanjoy Baruah. Multi-core cyclic executives for safety-critical systems. In *Dependable Software Engineering. Theories, Tools, and Applications - Third International Symposium, SETTA 2017, Changsha, China, October 23-25, 2017, Proceedings*, pages 94–109, 2017.

[8] AUTOSAR. Specification of autosar operating system, February 2013.

[9] Saoussen Anssi, Sara Tucci Piergiovanni, Stefan Kuntz, Sébastien Gérard, and François Terrier. Enabling scheduling analysis for AUTOSAR systems. In *ISORC 2011, Newport Beach, California, USA, 28-31 March 2011*, pages 152–159, 2011.

[10] Yunhui Peng, Yanhong Huang, Ting Su, and Jian Guo. Modeling and verification of AUTOSAR OS and EMS application. In *TASE 2013, 1-3 July 2013, Birmingham, UK*, pages 37–44, 2013.

[11] Yanhong Huang, João F. Ferreira, Guanhua He, Shengchao Qin, and Jifeng He. Deadline analysis of AUTOSAR OS periodic tasks in the presence of interrupts. In *ICFEM 2013, October 29 - November 1, 2013, Proceedings*, pages 165–181, 2013.

[12] Longfei Zhu, Peng Liu, Jianqi Shi, Zheng Wang, and Huibiao Zhu. A timing verification framework for AUTOSAR OS component development based on real-time maude. In *TASE 2013, 1-3 July 2013, Birmingham, UK*, pages 29–36, 2013.

[13] Qingling Zhao, Zonghua Gu, and Haibo Zeng. Design optimization for AUTOSAR models with preemption thresholds and mixed-criticality scheduling. *Journal of Systems Architecture - Embedded Systems Design*, 72:61–68, 2017.

[14] Leo Hatvani and Reinder J. Bril. Schedulability using native non-preemptive groups on an AUTOSAR/OSEK platform. In *ETFA 2015, Luxembourg, September 8-11, 2015*, pages 1–8, 2015.

[15] Hyunmin Yoon and Minsoo Ryu. Real-time priority assignment for autosar-based systems with time-driven synchronization. In *RACS 2014, Towson, Maryland, USA, October 5-8, 2014*, pages 297–302, 2014.

[16] Wenhao Wang, Fabrice Camut, and Benoit Miramond. Generation of schedule tables on multi-core systems for AUTOSAR applications. In *(DASIP), Rennes, France, October 12-14, 2016*, pages 191–198, 2016.

[17] INFINEON. Tc1782 32-bit single-chip microcontroller. Technical report, Infineon Technologies, 2011.

[18] OSEK. Osek/vdx operating system specification 2.2.3, February 2005.

[19] Michel Goossens, S. P. Rahtz, Ross Moore, and Robert S. Sutor. *The Latex Web Companion: Integrating TEX, HTML, and XML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[20] Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[21] Martin Stigge and Wang Yi. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems*, 51(6):639–674, 2015.

[22] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*, pages 201–209, 1990.