An engineering approach to synchronization based on overrun for compositional real-time systems

Citation for published version (APA):

Keskin, U., Heuvel, van den, M. M. H. P., Bril, R. J., Lukkien, J. J., Behnam, M., & Nolte, T. (2011). An engineering approach to synchronization based on overrun for compositional real-time systems. In *Proceedings of the 6th IEEE International Symposium on Industrial Embedded Systems (SIES 2011, Vasteras, Sweden, June 15-17, 2011)* (pp. 274-283). Institute of Electrical and Electronics Engineers.
<https://doi.org/10.1109/SIES.2011.5953671>

DOI:

[10.1109/SIES.2011.5953671](https://doi.org/10.1109/SIES.2011.5953671)

Document status and date:

Published: 01/01/2011

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

An engineering approach to synchronization based on overrun for compositional real-time systems

Uğur Keskin, Martijn M.H.P. van den Heuvel, Reinder J. Bril and Johan J. Lukkien

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

Moris Behnam and Thomas Nolte
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden

Abstract—Hierarchical scheduling frameworks (HSFs) provide means for composing complex real-time systems from well-defined independently developed and analyzed subsystems. To support shared logical resources requiring mutual exclusive access in two-level HSFs, overrun without payback has been proposed as a mechanism to prevent budget depletion during resource access arbitrated by the stack resource policy (SRP). In this paper, we revisit the *global schedulability analysis* of synchronization protocols based on SRP and overrun without payback for fixed-priority scheduled HSFs. We derive a new global schedulability analysis based on the observation that the overrun budget is merely meant to prevent budget depletion during global resource access. The deadline of a subsystem therefore only needs to hold for its normal budget rather than the sum of the normal and overrun budget. Our novel analysis is considerably simpler than an earlier, initially improved analysis, which improved both the original *local* and *global schedulability analyses*. We evaluate the new analysis based on an extensive simulation study and compare the results with the existing analysis. Our simplified analysis does not significantly affect schedulability compared to the initially improved analysis. It is therefore proposed as a preferable engineering approach to synchronization protocols for compositional real-time systems. We accordingly present the implementation of our improvement in an OSEK-compliant real-time operating system to sketch its applicability in today's industrial automotive standards. Both implementation and run-time overheads are discussed providing measured results¹.

I. INTRODUCTION

Hierarchical scheduling frameworks (HSFs) have been introduced to support hierarchical processor sharing among applications under different scheduling services [1]. An HSF can be represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g. tasks) and resources are allocated from a parent node to its children nodes. The HSF provides means for decomposing a complex system into well-defined parts called *subsystems*. It essentially provides a mechanism for timing-predictable *composition* of coarse-grained subsystems. These subsystems can be independently developed, analyzed

and tested. Temporal isolation between subsystems is provided through budgets which are allocated to subsystems.

Looking at existing industrial real-time systems, fixed-priority preemptive scheduling (FPPS) is the de facto standard of task scheduling, hence we focus on an HSF with support for FPPS within a subsystem. Having such support will simplify migration to and integration of existing legacy applications into the HSF. Our current research efforts are directed towards the conception and realization of a two-level HSF that is based on (i) FPPS for both *global scheduling* of budgets allocated to subsystems and *local scheduling* of tasks within a subsystem, (ii) the periodic resource model [1] for budgets, and (iii) the Stack Resource Policy (SRP) [2] for both local and global resource sharing.

Subsystems may share global logical resources requiring mutually exclusive access. An HSF without corresponding synchronization support is not realistic, since tasks in subsystems may for example use operating system services, memory mapped devices and shared communication devices. If a task that accesses a global shared resource is suspended during its execution due to the exhaustion of the corresponding budget, excessive blocking periods can occur which may hamper the correct timeliness of other subsystems [3]. We consider the *overrun without payback* mechanism [4], [5] to prevent depletion of a budget during global resource access by temporarily increasing the budget with a statically determined amount for the duration of that access. The term *without payback* means that the additional amount of budget does not have to be paid back during the next budget period. To distinguish this additional amount of budget from a *normal budget*, we will use the term *overrun budget*.

The global schedulability analysis in [4] is based on the assumption that the overrun budget is smaller than the normal budget. Moreover, it is stated that for well-constrained applications, the global resource access time will typically be much smaller than the normal budget. In this paper, we allow the overrun budget to become substantial compared to (or even larger than) the normal budget. This is motivated by the observation that arbitrary preemptions of tasks and subsystems

¹The work in this paper is supported by the Dutch HTAS-VERIFIED project, see <http://www.htas.nl/index.php?pid=154>.

may cause significant runtime overhead and high fluctuations in execution times, in particular due to architecture-related preemption costs [6], such as cache misses and pipeline flushes. Scheduling techniques aiming at a reduction of these preemption costs, such as fixed-priority scheduling with deferred preemption (FPDS) [7], may therefore significantly increase blocking times and overrun budgets. Our proposed analysis especially improves for relatively large overruns. Moreover, as large extents of embedded systems are resource constrained, a tight analysis is instrumental in a successful deployment of HSF techniques in real applications. Tighter local and global schedulability analysis has therefore been presented in [8] compared to the original analysis in [4], [5]. These improvements come at the cost of a significant increase in the analytical complexity. This paper reduces the complexity of the analysis in [8] without increasing its pessimism.

A. Contributions

We present a novel global schedulability analysis of synchronization protocols based on SRP and overrun without payback. Our analysis is based on the observation that the overrun budget is merely meant to prevent budget depletion during global resource access, and the deadline of a subsystem therefore only needs to hold for its normal budget rather than the sum of the normal and overrun budget. Our novel global analysis is considerably simpler than the initially improved analysis presented in [8]. Moreover, the results of our global analysis are at least as good as the global analysis presented in [8]. We illustrate this by means of an example. The improvement of the local analysis presented in [8] no longer applies to our novel analysis, however. We therefore evaluate the improvements of our novel analysis compared to the initially improved analysis by exploring the system load [5] in a simulation study. Finally, we evaluate an implementation of the overrun mechanism in an OSEK-compliant operating system that adheres to the new schedulability analysis. This marginally affects the implementation complexity and runtime overheads compared to our implementation in [9] which complies to both the original and initially improved analysis.

B. Overview

The remainder of this paper is as follows. Section II describes related work. Section III presents our real-time scheduling model. Section IV recapitulates the existing global and local schedulability analysis, i.e. both the original [4] and initially improved analysis [8]. Section V presents our new global schedulability analysis. Section VI compares our new analysis with the existing analysis by means of a simulation study. Section VII presents the implementation of the new overrun mechanism in an OSEK-compliant real-time operating system and its corresponding evaluation. Finally, Section VIII concludes this paper.

II. RELATED WORK

The increasing complexity of embedded real-time systems led to a growing attention for hierarchical scheduling of real-time systems [1], [10], [11], [12], [13]. Deng and Liu [10]

proposed a two-level HSF for open systems, where subsystems may be independently developed and validated. The corresponding schedulability analysis for two-level HSFs have been presented in [12] for FPPS and in [13] for earliest-deadline-first (EDF) global schedulers. Shin and Lee [1] proposed the periodic resource model to specify guaranteed periodic processor allocations to subsystems. Easwaran et al. [14] proposed the explicit-deadline periodic (EDP) resource model, which extends the periodic resource model by explicitly distinguishing a relative deadline for the allocation time of budgets.

For synchronization protocols in FPPS-based HSFs, two mechanisms have been presented to prevent budget depletion during global resource access, i.e. *overrun (with payback and without payback)* [4] and *self-blocking* [15]. Recently, both mechanisms have been analytically compared with respect to their impact on the total system load for various subsystem parameters [16]. The performance of each mechanism heavily depends on these chosen parameters.

Overrun with payback was first introduced in the context of aperiodic servers in [3]. This mechanism was later re-used for a synchronization protocol in the context of two-level HSFs in [4] and complemented with a variant *without payback*. Although the analysis presented in [4] does not integrate in HSFs due to the lacking support for independent analysis of subsystems, this limitation is lifted in [5], [17].

In [8] it is shown that the existing local and global schedulability analysis in [5] is pessimistic. The global analytical improvement presented in [8] is based on the observation that during an overrun, higher priority subsystems may experience limited preemptiveness. In addition, [8] improved on the local schedulability analysis by using the EDP resource model instead of the periodic resource model.

III. REAL-TIME SCHEDULING MODEL

We consider a two-level hierarchical FPPS model using the periodic resource model to specify guaranteed processor allocations to tasks of subsystems. Because the focus of this paper is on synchronization protocols for *global* logical resources, we do not consider local logical resources. We use a synchronization protocol for mutual exclusive resource access based on SRP and overrun without payback.

A. System model

A system Sys contains a set $\mathcal{R}$ of M global logical resources² $R_1, R_2, \dots, R_M$, a set $\mathcal{S}$ of N subsystems $S_1, S_2, \dots, S_N$, a set $\mathcal{B}$ of N budgets for which we assume a periodic resource model [1], and a single processor. Each subsystem S_s has a dedicated budget associated to it. In the remainder of this paper, we leave budgets implicit, i.e. the timing characteristics of budgets are taken care of in the description of subsystems. Each subsystem S_s therefore generates an infinite sequence of jobs τ_{sk} . Subsystems are scheduled by means of FPPS and have fixed, unique priorities. For notational convenience, we assume

²Non-preemptive executions of (i) regions of code to prevent architecture related preemption costs and (ii) operating-system services are also treated as access to so-called *pseudo* resources [18].

that subsystems are given in order of decreasing priorities, i.e. S_1 has highest priority and S_N has lowest priority.

B. Subsystem model

Each subsystem S_s contains a set $\mathcal{T}_s$ of n_s periodic tasks $\tau_1, \dots, \tau_{n_s}$ with fixed, unique priorities, which are scheduled by means of FPPS. For notational convenience, we assume that tasks are given in order of decreasing priorities, i.e. τ_1 has highest priority and τ_{n_s} has lowest priority. The set $\mathcal{R}_s$ denotes the subset of M_s global resources accessed by subsystem S_s . The maximum time that a subsystem S_s executes while accessing resource $R_l \in \mathcal{R}$ is denoted by X_{sl} , where $X_{sl} \in \mathbb{R}^+ \cup \{0\}$ and $X_{sl} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$. The timing characteristics of S_s are specified by means of a triple $\langle P_s, Q_s, X_s \rangle$, where $P_s \in \mathbb{R}^+$ denotes its (budget) period, $Q_s \in \mathbb{R}^+$ its (normal) budget, and X_s the set of maximum execution access times of S_s to global resources. The maximum value in X_s is denoted by X_s .

C. Task model

The timing characteristics of a task $\tau_{si} \in \mathcal{T}_s$ are specified by means of a quartet $\langle T_{si}, C_{si}, D_{si}, C_{si} \rangle$, where $T_{si} \in \mathbb{R}^+$ denotes its minimum inter-arrival time, $C_{si} \in \mathbb{R}^+$ its worst-case computation time, $D_{si} \in \mathbb{R}^+$ its (relative) deadline, C_{si} a set of maximum execution times of τ_{si} to global resources, where $C_{si} \leq D_{si} \leq T_{si}$. The set $\mathcal{R}_{si}$ denotes the subset of $\mathcal{R}_s$ accessed by task τ_{si} . The maximum time that a task τ_{si} executes while accessing resource $R_l \in \mathcal{R}$ is denoted by c_{sil} , where $c_{sil} \in \mathbb{R}^+ \cup \{0\}$, $C_{si} \geq c_{sil}$, and $c_{sil} > 0 \Leftrightarrow R_l \in \mathcal{R}_{si}$.

D. Resource model

The *processor supply* refers to the amount of processor allocation that a virtual processor can provide. The supply bound function $\text{sbf}_{\Omega_s}(t)$ of the EDP resource model $\Omega_s(P_s, Q_s, \Delta_s)$ that computes the minimum possible processor supply for every interval length t to a subsystem S_s is given by [14]:

$$\text{sbf}_{\Omega_s}(t) = \begin{cases} t - (k+1)(P_s - Q_s) + (P_s - \Delta_s) & \text{if } t \in V^{(k)} \\ (k-1)Q_s & \text{otherwise,} \end{cases} \quad (1)$$

where $k = \max\left(\left\lceil \frac{t - (\Delta_s - Q_s)}{P_s} \right\rceil, 1\right)$ and $V^{(k)}$ denotes an interval $[kP_s + \Delta_s - 2Q_s, kP_s + \Delta_s - Q_s]$. The supply bound function $\text{sbf}_{\Gamma_s}(t)$ of the periodic resource model $\Gamma_s(P_s, Q_s)$ is a special case of (1), i.e. with $\Delta_s = P_s$.

E. Synchronization protocol

Overrun without payback prevents depletion of a budget of a subsystem S_s during access to a global resource R_l by temporarily increasing the budget of S_s with X_{sl} , i.e. the maximum time that S_s executes while accessing R_l . To be able to use SRP in an HSF for synchronizing global resources, its associated ceiling terms need to be extended.

1) *Resource ceiling*: With every global resource R_l , two types of resource ceilings are associated; an *external* resource ceiling RC_l for global scheduling of budgets and an *internal*

resource ceiling rc_{sl} for local scheduling of tasks. According to SRP, these ceilings are defined as

$$RC_l = \min(N, \min\{s \mid R_l \in \mathcal{R}_s\}), \quad (2)$$

$$rc_{sl} = \min(n_s, \min\{i \mid c_{sil} > 0\}). \quad (3)$$

We use the outermost min in (2) and (3) to define RC_l and rc_{sl} also in those situations where no subsystem uses R_l and no task of $\mathcal{T}_s$ uses R_l , respectively.

2) *System/subsystem ceiling*: The system/subsystem ceilings are dynamic parameters that change during the execution. The system/subsystem ceiling is equal to the highest (numerically smallest) external/internal resource ceiling of a currently locked resource in the system/subsystem.

Under SRP, a task τ_{si} can only preempt the currently executing task τ_{sj} if the priority of τ_{si} is higher (i.e. the index i is lower) than the subsystem ceiling of S_s . A similar condition for preemption holds for subsystems.

3) *Concluding remarks*: The maximum time X_{sl} that S_s executes while accessing R_l can be reduced by assigning a value to rc_{sl} that is *smaller* than the value according to SRP. For HSRP [4], the internal resource ceiling is therefore set to the highest priority, i.e. $rc_{sl}^{\text{HSRP}} = 1$. Decreasing rc_{sl} may cause a subsystem to become unfeasible for a given budget [19], however, because the tasks with a priority higher than the old ceiling and at most equal to the new ceiling may no longer be feasible. The results in this paper apply for any internal resource ceiling rc'_{sl} where $rc_{sl} \geq rc'_{sl} \geq rc_{sl}^{\text{HSRP}} = 1$.

IV. EXISTING SCHEDULABILITY ANALYSIS

This section briefly recapitulates the original analysis [4] and the initially improved schedulability analysis [8].

A. Original schedulability analysis

Although the global schedulability analysis presented in [5], [16] looks different, it is based on the analysis described in [4] and therefore yields the same result.

1) *Global analysis*: The worst-case response time WR_s of subsystem S_s is given by the smallest $x \in \mathbb{R}^+$ satisfying

$$x = B_s + (Q_s + X_s) + \sum_{t < s} \left\lceil \frac{x}{P_t} \right\rceil (Q_t + X_t), \quad (4)$$

where B_s is the maximum blocking time of S_s by lower priority subsystems, i.e.

$$B_s = \max(0, \max\{X_{tl} \mid t > s \wedge X_{tl} > 0 \wedge RC_l \leq s\}). \quad (5)$$

We use the outermost max in (5) to define B_s also in those situations where the set of values of the innermost max is empty. To calculate the worst-case response time WR_s , we use an iterative procedure based on recurrence relationships, starting with a lower bound, e.g. $B_s + \sum_{t \leq s} (Q_t + X_t)$. The condition for global schedulability is given by

$$\forall_{1 \leq s \leq N} WR_s \leq P_s. \quad (6)$$

The global analysis for subsystems is similar to basic analysis for tasks under FPPS with resource sharing [20]. For the global analysis, the period P_s of a subsystem S_s serves as a deadline

for the *total* budget $Q_s + X_s$ of S_s . Similarly, the interference of higher priority subsystems S_t is based on the sum $Q_t + X_t$. A superscript P will be used to refer to this basic analysis for subsystems, e.g. WR_s^P .

In the sequel, we are not only interested in the worst-case response time WR_s of a subsystem S_s for particular values of B_s , Q_s , and X_s , but in the value as a function of the sum of these three values. We will therefore use a functional notation when needed, e.g. $WR_s(B_s + Q_s + X_s)$.

2) *Local analysis*: The existing condition for the local schedulability of tasks within a subsystem S_s [5] is given by

$$\forall_{1 \leq i \leq n_s} \exists_{0 < t \leq D_{si}} b_{si} + C_{si} + \sum_{j < i} \left\lceil \frac{t}{T_{sj}} \right\rceil \cdot C_{sj} \leq \text{sbfb}_{\Gamma_s}(t), \quad (7)$$

where b_{si} is the maximum blocking time of τ_{si} by lower priority tasks, i.e.

$$b_{si} = \max(0, \max\{c_{sjl} \mid j > i \wedge c_{sjl} > 0 \wedge rc_{sl} \leq i\}), \quad (8)$$

and $\text{sbfb}_{\Gamma_s}(t)$ is the supply bound function of the periodic resource model $\Gamma_s(P_s, Q_s)$ for the subsystem S_s under consideration. We use the outermost max in (8) to define b_{si} also in those situations where the set of values of the innermost max is empty.

The value for X_{sl} of subsystem S_s is given by

$$X_{sl} = \max_{1 \leq i \leq n_s} X_{sil}, \quad (9)$$

where X_{sil} denotes the maximum time that S_s executes while task τ_{si} accesses resource $R_l \in \mathcal{R}_s$, with $X_{sil} \in \mathbb{R}^+ \cup \{0\}$ and $X_{sil} > 0 \Leftrightarrow c_{sil} > 0$. For $c_{sil} > 0$, X_{sil} is given by [5]

$$X_{sil} = c_{sil} + \sum_{j < rc_{sl}} C_{sj}. \quad (10)$$

B. Initially improved analysis

The schedulability analysis presented in [8] improves on both the initial global and local analysis in [4], [5].

1) *Global analysis*: Similar to the original analysis, the period P_s of a subsystem S_s serves as a deadline for the total budget $Q_s + X_s$. The improvements presented in [8] are based on two observations: (i) while S_s is accessing R_l using X_s , it can only be preempted by subsystems with a higher priority than RC_l , and (ii) blocking starts *before* the consumption of the overrun budget X_s starts. Due to (i), the improved global analysis is similar to the analysis for FPDS [7] and FPPS with preemption thresholds [21] in the sense that all jobs in a so-called *level-s active period* have to be considered to determine the worst-case response time WR_s of subsystem S_s . Unlike the analysis described in [7], [21], subsystems S_{s-1} till S_{RC_l} cannot preempt S_s at the finalization time of Q_s when S_s is accessing R_l , which is an immediate consequence of (ii). Finally, when a subsystem S_s uses multiple global resources, i.e. $M_s > 1$, the schedulability of S_s potentially needs to be determined for each of those resources; see [8] for more details. These improvements increase the complexity of the analysis significantly compared to [4], [5].

2) *Local analysis*: The improved local analysis is based on the observation that when a system is feasible from a global scheduling perspective, i.e. the deadline P_s of subsystem S_s is met for the total budget $Q_s + X_s$, the latest finalization time of Q_s is guaranteed to be at least X_s before the next activation of S_s . Hence, the supply bound function $\text{sbfb}_{\Omega}(t)$ of the EDP resource model $\Omega_s(P_s, Q_s, \Delta_s)$ for overrun without payback can be used rather than $\text{sbfb}_{\Gamma_s}(t)$ of the periodic resource model $\Gamma_s(P_s, Q_s)$ in (7), where $\Delta_s = P_s - X_s$. Because $X_s \geq 0$ for all subsystems (by definition), $\text{sbfb}_{\Gamma_s}(t) \leq \text{sbfb}_{\Omega_s}(t)$ for all subsystems. As a result, a subsystem may be schedulable according to the local analysis based on $\text{sbfb}_{\Omega_s}(t)$, but not be schedulable based on $\text{sbfb}_{\Gamma_s}(t)$.

V. THE NEW GLOBAL SCHEDULABILITY ANALYSIS

The overrun budget X_s is only meant to prevent budget depletion of subsystem S_s during global resource access. As a result, the period P_s of S_s only needs to serve as a deadline for the normal budget Q_s of S_s and not for its total budget $Q_s + X_s$. This allows for a further improvement of the global analysis compared to the initially improved analysis [8]. It also requires us to revert to the original schedulability analysis for the local analysis, because P_s does not need to be met by X_s , and $P_s - X_s$ therefore does not need to be met by Q_s .

In the remainder of this section, we first recapitulate the notion of a level- s active period as defined in [7]. Next, we present analysis for the worst-case response time WR_s^Q of the normal budget Q_s of subsystem S_s . We subsequently present an example illustrating the improvement of our novel global analysis compared to the original and the initially improved analysis. This section is concluded with a discussion on the strong and weak points of our novel analysis compared to the initially improved analysis.

A. Level- s active period

An *active interval* of a job of a subsystem is defined as the time span between the activation time of that job and its finalization time. A *level- s active period* is a *smallest* interval that only contains entire active intervals of jobs of subsystem S_s and jobs of subsystems with a higher priority than S_s . The worst-case length WL_s of a level- s active period is found when the level- s active period starts at a so-called *ϵ -critical instant*, i.e. when S_s has a simultaneous release with all higher priority subsystems and a lower priority subsystem with a maximum blocking time B_s for S_s starts its access to the associated global shared resource an infinitesimal time ϵ before that simultaneous release. The worst-case length WL_s is given by the smallest $x \in \mathbb{R}^+$ that satisfies

$$x = B_s + \sum_{t \leq s} \left\lceil \frac{x}{P_t} \right\rceil (Q_t + X_t). \quad (11)$$

To calculate WL_s , we can use an iterative procedure based on recurrence relationships, starting with a lower bound, e.g. $B_s + \sum_{t \leq s} (Q_t + X_t)$. The maximum number wl_s of jobs of S_s in a level- s active period is given by

$$wl_s = \left\lceil \frac{WL_s}{P_s} \right\rceil. \quad (12)$$

Fig. 1. Global schedulability volumes based on (a) the original analysis [4], (b) the initial improvements [8], and (c) our new analysis.

B. Worst-case response time

The worst-case response time WR_s^Q of the normal budget Q_s of subsystem S_s is given by the largest response time of Q_s of the jobs of S_s in a level- s active period that starts at an ϵ -critical instant, i.e.

$$WR_s^Q = \max_{0 \leq k < wl_s} WR_{sk}^Q, \quad (13)$$

where WR_{sk}^Q denotes the worst-case response time of Q_s of job ι_{sk} for subsystem S_s . To determine WR_{sk}^Q we have to consider up to three suprema. First, the sequence of jobs ι_{s0} till ι_{sk} experience a blocking $B_s \geq 0$ by lower priority subsystems in the worst-case situation. Similar to FPDS [7], the worst-case blocking is a supremum for $B_s > 0$ rather than a maximum. Second, the jobs ι_{s0} till $\iota_{s,k-1}$ need their overrun budget X_s to access global resources. Because the access to a global resource starts during the execution of the normal budget, the actual amount of overrun budget that is used is a supremum rather than a maximum. Finally, in the worst-case scenario the access to the global resource also starts *as late as possible* during the execution of job ι_{sk} . This maximizes the interference of higher priority subsystems, and also gives rise to a supremum rather than a maximum. The worst-case response time WR_{sk}^Q can therefore be described as

$$WR_{sk}^Q = \lim_{Q \uparrow} \lim_{X \uparrow} \lim_{B \uparrow} WR_s^P(B + k(Q_s + X) + Q) - kP_s, \quad (14)$$

where WR_s^P is the worst-case response time of a fictive subsystem S'_s with a period $P'_s = (k+1)P_s$, a normal budget $Q'_s = k(Q_s + X) + Q$ and a maximum blocking time B , and kP_s represents the activation of job ι_{sk} relative to the start of the level- s active period. Using the following equation from [7]

$$\lim_{x \uparrow C} WR_i^P(x) = WR_i^P(C) \quad (15)$$

we derive

$$WR_{sk}^Q = WR_s^P(B_s + (k+1)Q_s + kX_s) - kP_s. \quad (16)$$

C. An example

For illustration purposes, we will use an example system Sys_1 containing three subsystems S_1 , S_2 and S_3 sharing a global resource R_1 . The characteristics $\langle P_s, Q_s, X_s \rangle$ of the subsystems are given in Table I.

TABLE I
SUBSYSTEM CHARACTERISTICS OF Sys_1 .

subsystem	P_s	$Q_s + X_s$
S_1	6	2
S_2	8	$Q_2 + X_2$
S_3	10	$1 + X_3$

Figure 1 illustrates the global schedulability of (a) the original [4], (b) the initially improved [8] and (c) our new analysis by means of feasibility volumes for the example system Sys_1 . It is clear that the initial improvements in [8] (Figure 1(b)) significantly extends the feasibility volume compared to the original analysis (Figure 1(a)). Our novel analysis only slightly extends the volume, however, as shown in Figure 1(c). We take a closer look at the latter improvement by means of a specific example.

Example: Sys_1 with $Q_2 = 2.0$, $X_2 = 1.0$ and $X_3 = 1.8$.

We determine WR_3^Q using the analysis described above; see also Figure 2. Because S_3 is the lowest priority subsystem, $B_3 = 0$. We first determine wl_3 using (11) and (12), and find $WL_3 = 96$ and $wl_3 = \lceil WL_3 / P_3 \rceil = \lceil 96 / 10 \rceil = 10$. For space considerations, we will only determine $WR_{3,3}^Q$, which is the maximum value for those 10 jobs. We get $WR_{3,3}^Q = WR_s^P(4 + 5.4) - 3 \cdot 10 = 38.4 - 30 = 8.4$ using (16). Finally, using (13) we find $WR_3^Q = \max_{0 \leq k < 10} WR_{3,k}^Q = 8.4$.

The worst-case response time of the total budget $Q_3 + X_3$ of S_3 as determined by the initial improvements in [8] is equal to 10.2, which is larger than the deadline $P_3 = 10$ of S_3 ; see also Figure 2. The example is therefore not schedulable according to the global analysis in [8], but can be scheduled using our new analysis.

D. Concluding remarks

Our novel global analysis treats the period P_s as a deadline for Q_s , rather than $Q_s + X_s$ as is done in the initially improved

Fig. 2. The timeline for S_{sys1} , with $Q_2 = 2.0$, $X_2 = 1.0$ and $X_3 = 1.8$, illustrates that the overrun budget X_3 of job $1_{3,3}$ of S_3 completes *after* the activation of job $1_{3,4}$. The analysis in [8] therefore concludes that subsystem S_3 misses a deadline at time 40, whereas our analysis deems this example schedulable.

analysis in [8]. This has three consequences. Firstly, the improvement of the local analysis of the initially improved analysis no longer applies to our novel analysis. Secondly, the results of our novel global analysis are at least as good as the initially improved analysis. Finally, major parts contributing to the complexity of the global analysis introduced in [8] have been removed. In particular, our novel global analysis does not address the preemptions of the overrun budget for every job of S_s nor needs to separately determine the schedulability of S_s for each global resource $R_l \in \mathcal{R}_s$ in case S_s is using multiple resources. However, due to the limited preemption of the overrun budget X_s , we may still have to consider more than one job, i.e. all jobs in the level- s active period.

The reduction of complexity of our novel global analysis compared to the initially improved analysis is considered to be particularly useful for dynamic systems, where subsystems can enter or leave the system during runtime and the global analysis needs to be repeated as part of the admission test.

VI. EVALUATION

This section evaluates our novel schedulability analysis (NSA) for the overrun mechanism, based on a subsystem's budget with respect to processor resources. We compare NSA with the initially improved schedulability analysis (IISA) presented in [8] and with the original schedulability analysis (OSA) of the overrun mechanism using the notion of system load [5]. The system load provides an indication of the system's processor requirements. Our comparison is carried out by means of simulation experiments to show the performance of NSA relative to alternative approaches.

First, we briefly explain the notion of system load. Secondly, we show how it is adapted for NSA.

a) *System load*: The system load is defined as a quantitative measure to represent the minimum amount of processor allocations which guarantee the global schedulability of a system S_{sys} . OSA calculates the system load, $load_{S_{sys}}$, by

$$load_{S_{sys}} = \max_{\forall S_s \in \mathcal{S}} \{\alpha_s\}, \quad (17)$$

where

$$\alpha_s = \min_{0 < x \leq P_s} \left\{ \frac{RBF_s(x)}{x} \mid RBF_s(x) \leq x \right\} \quad (18)$$

and the request bound function, $RBF_s(x)$, of a subsystem S_s is defined by the right-hand side of (4). Note that x can be selected from a finite set of scheduling points [22] and that α_s is the smallest fraction of the processor resource that is required to schedule a subsystem S_s . This satisfies the schedulability condition presented in Section IV-A1, assuming a global resource-supply of $\alpha_s x$. One can think of system load as decreasing the speed of the processor by the factor $load_{S_{sys}}$. This increases a subsystem's normal budget, overrun budget, and blocking times by a factor $1/load_{S_{sys}}$.

b) *Evaluating system load*: Evaluating the system load for IISA and NSA is more complex than for OSA, because it has more than one response-time equation to determine the global schedulability of subsystems (see Section V). We therefore cannot apply the OSA approach to evaluate the system load for IISA and NSA.

We solve this problem by using a binary search algorithm, such that the system load is selected by the search algorithm and corresponding system schedulability is checked [8]. We therefore multiply the normal budgets, maximum overrun budgets, and blocking times of all subsystems in equations (11) - (16) by a factor $1/load_{S_{sys}}$. If the system is schedulable, then the algorithm selects a lower system load and try again. If the system is unschedulable, then the algorithm selects a higher system load. The algorithm terminates if the selected system load $load_{S_{sys}} > 1$ and the system is unschedulable, or when the difference between the previous and the current system load is less than a given acceptance limit. Because we use a binary search algorithm for both NSA and IISA, the complexity of evaluating the system load is significantly higher compared with OSA. However, note that we merely use the system load for comparison purposes. Hence, it has no relationship with the complexity of the schedulability analysis.

The efficiency of NSA is measured by the required system load of a subsystem in order to be schedulable relative to IISA and OSA. Given that NSA excludes overrun budgets from the response-time analysis and that the analysis is only based on a subsystem's normal budget, NSA requires less system load to guarantee the schedulability of systems compared to OSA. This is not necessarily true for IISA, because IISA also improves the local schedulability analysis, i.e. the local analysis may decrease a subsystem's normal budget.

A. Simulation setting

Our simulation applies the analysis of NSA, IISA and OSA on 1000 randomly generated systems. Initially, we assume that each system comprises 5 subsystems and each subsystem contains 4 tasks. We assume only one global shared resource and all subsystems share this resource. Two tasks in each subsystem access this global shared resources.

As shown in [8], the gain for IISA compared to OSA is maximized when critical sections are executed with local and global preemptions disabled. In line with this observation and for simplicity, we assume that the internal resource ceilings of global shared resources are equal to the highest task priority in each subsystem (i.e., $rc_{sl} = 1$), and $T_{si} = D_{si}$ for all tasks. For

Fig. 3. The distribution of randomly generated subsystems based on their system load for Study 1 with (a) $CS_s = 2$ and (b) $CS_s = 4$.

each simulation study a new set of 1000 systems is generated and the following settings are changed:

1) *Critical-section execution time (CS_s)*: It specifies the maximum time that a task may execute while accessing a global shared resource. Changing this parameter does not require to generate new systems, since this does not affect other task parameters, as we will show later. Although the critical-section execution time is given as an input parameter, its value cannot be greater than the execution time of a task. The actual critical-section execution time c_{sil} is therefore set to the minimum value of the task's execution time and the given input parameter CS_s , i.e., $c_{sil} = \min(CS_s, C_{si})$.

2) *The subsystem period (P_s) and task periods (T_{si})*: We decrease the ratio between the critical-section execution times and a subsystem's period, i.e. CS_s/P_s , by increasing subsystem and task periods. These periods are specified as a range with a lower and upper bound. The simulation program randomly generates subsystem and task periods within this range following a uniform distribution.

3) *Number of subsystems (N)*: increase the number of subsystems in the system.

4) *System utilization (U^{Sys})*: The sum of the utilization of all tasks in the system is fixed to a specified value, U^{Sys} . The given system utilization is randomly divided among the subsystems. The assigned utilization to each subsystem is in turn randomly divided to its tasks. Since the values of task periods are generated within a specified interval, their execution times are derived from the task's utilization. All randomized system parameters are generated following uniform distributions.

B. Simulation results

We have performed four different simulation studies similar to the studies in [8]. The range of the random values is selected so that it highlights some properties of the new analysis:

- **Study 1** specifies critical-section execution times $CS_s \in \{2, 4\}$, task periods $T_{si} \in [140, 1000]$, subsystem periods $P_s \in [40, 70]$, $U^{Sys} = 20\%$ and $N = 5$.
- **Study 2** increases the range of the subsystem periods P_s and task periods T_{si} (compared to Study 1) to $P_s \in [100, 200]$ and $T_{si} \in [400, 1000]$ with $CS_s = 2$.

- **Study 3** changes the number of subsystems (compared to Study 1) to $N = 8$ with $CS_s = 2$.
- **Study 4** changes the system utilization (compared to Study 1) to $U^{Sys} = 30\%$ with $CS_s = 2$.

Figure 3(a) and Figure 3(b) show the results of **Study 1** for the case of $CS_s = 2$ and $CS_s = 4$ using OSA, IISA and NSA. These figures show the distribution of all randomly generated subsystems that have a system load within the ranges shown in the x-axis. The lines that connect points are only used for illustration purposes. Figure 4 shows the results of **Study 2**. Figure 5 and Figure 6 show the results of **Study 3** and **Study 4**.

C. Evaluation

Given our simulation results we can conclude that neither IISA nor NSA is superior. In the remainder of this section we therefore focus on the comparison of NSA to OSA.

1) *Increasing critical-section execution times (CS_s)*: Comparing the results of NSA and OSA and given that the NSA results are almost the same as the results of IISA, the same conclusion made in [8] is also valid for this case. When critical-section execution times CS_s are increased, NSA achieves better results than OSA, see Figure 3(a) and Figure 3(b). In other words, if the ratio CS_s/P_s is relatively high (i.e., X_s/P_s), then NSA performs significantly better than OSA.

2) *Increasing the subsystem period (P_s) and task periods (T_{si}) ranges*: In **Study 2**, we decrease the ratio CS_s/P_s by increasing the range of the subsystem periods. In this case, the improvement that NSA can achieve is less than in **Study 1**, because $X_s/load_{sys}$ becomes less significant within the subsystem period P_s (compare Figure 3(a) with Figure 4).

3) *Increase the number of subsystems (N)*: In **Study 3**, we investigate the effect of increasing the number of subsystems compared to **Study 1**. The results are shown in Figure 5. We can see that increasing N decreases the improvement of NSA over OSA, because increasing the number of subsystems will increase the interference of the higher priority subsystems. In turn, this decreases our improvement.

4) *Increase the system utilization (U^{Sys})*: Finally, in **Study 4** we investigate the effect of increasing the system utilization on the performance of NSA compared to **Study 1**.

When comparing the results of Figure 6 with Figure 3(a), we can see that increasing the value of the system utilization, U^{Sys} , decreases the improvement that NSA can achieve over OSA. The reason for this is that increasing the value of U^{Sys} increases the normal budget of all subsystems in the system. This also increases the interference of the higher priority subsystems, see (11) - (16), and will therefore limit the potential improvement of NSA compared to OSA.

5) *Concluding remarks:* Looking at the results of all figures, we can see that the system load required by both IISA and NSA is nearly the same. As explained previously, the improvement in the global analysis that NSA can achieve compared to IISA is limited by the improvement in the local analysis of IISA. For example, by increasing the critical-section execution times to $CS_s = 4$ in **Study 1**, one would expect that NSA provides better results than IISA, because an increasing CS_s also increases X_s and the deadline P_s only holds for Q_s for NSA rather than for $Q_s + X_s$ for IISA. This increased X_s is (partially) excluded from our new analysis. At the same time, however, such an increase of X_s also increases the improvement in the local analysis of IISA.

Changing other system parameters, e.g. subsystem periods, the number of subsystems and the subsystem utilization, has the same effect on both IISA and NSA as shown by **Study 2**. Note that we considered the case where IISA has its maximum gain compared to OSA, i.e. local and global preemptions are disabled while executing in a critical section. If we consider a setup where resource ceilings are configured such that (global) preemptions are allowed during the execution of a critical section, then the improvement of IISA will be limited compared to OSA. However, this does not affect the improvement that NSA can provide.

Fig. 4. Results of Study 2.

VII. MODIFIED OVERRUN IMPLEMENTATION

This section outlines our overrun implementation in an OSEK-compliant real-time operating system, $\mu C/OS-II$ [23]. The kernel is open source, extensively documented [24] and applied in many application domains, e.g. avionics, automotive, medical and consumer electronics. We first give an overview of prerequisite $\mu C/OS-II$ extensions. Next we present our protocol implementation, which builds on top of a two-level HSF with corresponding support for SRP [9]. We cannot

Fig. 5. Results of Study 3.

Fig. 6. Results of Study 4.

re-use our existing implementation of overrun [9], because our presented analysis possibly introduces a budget replenishment while a subsystem has remaining overrun budget.

A. Timed event management

Intrinsic to our reservation-based subsystem scheduler is timed-event management. This comprises timers to accommodate (i) periodic timers at the global level for budget replenishment of periodic servers and at the subsystem level to enforce minimal inter-arrivals of sporadic task activations and (ii) virtual timers to track a subsystem's budget.

When these event timers expire, e.g. budget depletion and budget replenishment, their corresponding handlers are executed in the context of the timer interrupt service routine (ISR). We refer to [25] for a more extensive overview of such a timer-management module implemented and evaluated in $\mu C/OS-II$.

B. Two-level HSF with SRP support

For ease of presentation, we limit our implementation to idling periodic servers [26] to allocate budgets to subsystems. Extending $\mu C/OS-II$ with basic HSF support requires the identification and realization of the following concepts:

1) *Subsystems:* $\mu C/OS-II$ tasks are bundled in groups of sixteen to accommodate efficient FPPS [24]. A subsystem is therefore naturally represented by such a group.

2) *Periodic servers*: A realization of the idling periodic server is very similar to the implementation of a periodic task using our timed-event management. An idling server contains an idle task at the lowest, local priority, which is always ready to execute and cannot be blocked.

3) *Two-level SRP*: We extended the $\mu\text{C}/\text{OS-II}$ scheduler with SRP's notion of system and subsystem ceilings. We re-use our two-level SRP-implementation in [9] to maintain each of these subsystem and system ceilings by means of a stack data structure. The primitive *updateSubsystemCeiling* maintains the local ceiling stack. We define an SRP interface to access global resources and to maintain its corresponding data structure, i.e.:

- 1) void SRPMutexLock(Resource* r);
- 2) void SRPMutexUnlock(Resource* r);

After *SRPMutexUnlock* has reduced the system ceiling, it calls the scheduler. The values on top of the local/global ceiling stacks represent the current subsystem and system ceilings.

C. Protocol implementation

In many microkernels, including $\mu\text{C}/\text{OS-II}$, the only way for tasks to share data structures with ISRs is by means of disabling interrupts. We therefore assume that our synchronization primitives execute non-preemptively with interrupts disabled. In addition to the implementation of the lock and unlock operations, we need to adapt the budget-depletion and replenishment event handlers to cope with overrun. This requires to keep track of the number of resources locked within subsystem S_s and whether or not a server executes in its overrun budget X_s . The server data-structure is therefore extended with four fields for bookkeeping purposes, i.e. *lockedResourceCounter*, *inOverrun*, *replenishBudget* and X_s .

1) *Resource locking*: The lock operation is a straightforward two-level SRP-based lock. It first updates the locked resource counter and the subsystem's local ceiling to limit interference of tasks within the subsystem itself and subsequently updates the system ceiling with *SRPMutexLock*.

2) *Resource unlocking*: Unlocking a resource means that the subsystem/system ceiling must be updated and the SRP resource must be released. In case any overrun budget is consumed and no other global resource is locked within the same subsystem, we need to inform the scheduler that overrun has ended. However, if a replenishment has occurred during the overrun duration, i.e. $S_s.\text{replenishBudget} = \text{true}$, we need to execute a deferred replenishment of the subsystem's budget. The unlock operation in pseudo-code is:

Algorithm 1 void NSA_unlock(Resource* r);

```

1: updateSubsystemCeiling();
2:  $S_s.\text{lockedResourceCounter} - -$ ;
3: if  $S_s.\text{lockedResourceCounter} = 0$  and  $S_s.\text{inOverrun}$  then
4:   if  $S_s.\text{replenishBudget} = \text{true}$  then
5:      $S_s.\text{replenishBudget} \leftarrow \text{false}$ 
6:     setSubsystemBudget( $Q_s$ );
7:   else
8:     setSubsystemBudget(0);
9:   end if
10:  $S_s.\text{inOverrun} \leftarrow \text{false}$ ;
11: end if
12: SRPMutexUnlock(r);

```

The command *setSubsystemBudget(0)* performs two actions: (i) the server is blocked to prevent the scheduler from rescheduling the server, and (ii) the virtual timer, which tracks a subsystem's budget depletion, is canceled.

3) *Budget depletion*: We extend the event handler for budget depletion with the following rule: if any task within the subsystem holds a resource, then the budget is replenished with an amount X_{sl} and server inactivation is postponed. This requires to set a new virtual timer with the value X_{sl} .

4) *Budget replenishment*: For each periodic server an event handler is periodically executed to recharge its budget. If a replenishment timer P_s expires while subsystem S_s executes in its overrun budget, we cannot replenish budget Q_s , because any remaining overrun budget must be discarded when a subsystem releases its resources. In order to avoid multiple, expensive timer manipulations, a replenishment must be deferred until overrun ends, i.e. all resources $R_l \in \mathcal{R}_s$ are unlocked. We implement this by setting $S_s.\text{replenishBudget} \leftarrow \text{true}$.

D. Synchronization overheads

We recently created a port for $\mu\text{C}/\text{OS-II}$ to the OpenRISC platform [27] to experiment with the accompanying cycle-accurate simulator. The OpenRISC simulator allows software-performance evaluation via a cycle-count register. This profiling method may result in either longer or shorter measurements between two matching calls due to the pipelined OpenRISC architecture. Some instructions in the profiling method interleave better with the profiled code than others. The measurement accuracy is approximately 5 instructions.

1) *Time complexity*: Since it is important to know whether a real-time operating system behaves in a timewise predictable manner, we investigate the disabled interrupt regions caused by the execution of overrun primitives. Our synchronization primitives are independent of the number of servers and tasks in a system, but introduce overheads that interfere at the system level due to their required budget-timer manipulations. This makes our primitives more expensive than a straightforward two-level SRP implementation.

TABLE II
BEST-CASE (BC) AND WORST-CASE (WC) EXECUTION-TIMES FOR
OVERRUN MEASURED IN NUMBER OF PROCESSOR INSTRUCTIONS.

Event	SRP		Overrun [9]		Improved overrun	
	BC	WC	BC	WC	BC	WC
Lock	124	124	196	196	196	196
Unlock	106	106	196	725	196	735
Deplete	-	-	0	383	0	383
Replenish	-	-	0	0	0	15

Table II compares the execution times of SRP-based synchronization primitives. The best-case overhead is null in addition to the normal number of processor instructions that are spent to increase and decrease the subsystem and system ceilings. The worst-case overhead occurs at the start and end of an overrun situation. When the budget depletes while a subsystem has locked a resource, it is replenished with an overrun budget of X_s , which takes 383 instructions. Overrun completion can merely occur when a task unlocks a resource

while consuming overrun budget. The system overhead to update a subsystem's budget is 735 instructions, i.e. the test for replenishment in the unlock operation adds 10 instructions. Note that the worst-case execution time of an unlock operation and a budget depletion handler can never happen in the same subsystem period, i.e. the jitter caused by the synchronization primitives remains almost the same. These execution times of the primitives must be included in the system analysis by adding these to critical-section execution times, X_{sl} .

2) *Memory complexity*: The code sizes in bytes of the lock and unlock operations, i.e. 436 and 532 bytes, is higher than the size of plain SRP, i.e. 196 and 192 bytes. This includes two levels of SRP, the overrun mechanism and timer management. The size of the unlock operation is increased compared to our implementation in [9]. However, $\mu C/OS-II$'s priority-inheritance protocol has larger sized lock and unlock primitives, i.e. 924 and 400 bytes.

Moreover, each SRP resource has a data structure in (i) each subsystem that shares this resource and (ii) at the global level. These memory requirements are unchanged compared to our earlier two-level SRP implementation [9].

VIII. CONCLUSION

We revisited synchronization protocols based on SRP for two-level fixed-priority scheduled HSFs that prevent budget depletion by an overrun (without payback) mechanism. Whereas the original analysis in [4] is pessimistic, because it does not consider the limited preemptiveness of subsystems during overrun, the improved analysis in [8] is considerably more complicated. We simplified the global schedulability analysis based on the observation that the deadline for each subsystem only holds for its normal budget rather than the sum of its normal and overrun budget. This reduction of complexity compared to the initially improved analysis is particularly useful for dynamic systems where the global analysis is part of a subsystem's admission test.

Because neither the initial improved analysis nor our novel schedulability analysis is superior, we evaluated our new analysis based on an extensive simulation study. Both the initial improved analysis [8] and our novel analysis are especially beneficial when critical sections are relatively long compared to a subsystem's budget. This enables a tight analysis for HSFs in which a limited number of arbitrary preemptions can reduce architecture-related preemption costs.

Our presented analysis allows a subsystem to have remaining overrun when its normal budget replenishes. Hence, we cannot re-use earlier implementations of the overrun mechanism. We therefore implemented our new overrun mechanism in an OSEK-compliant real-time operating system and showed that its increase in complexity and run-time overheads is only marginal. Because our novel overrun mechanism comes with a tight, simplified analysis and its implementation costs are minor, it provides a promising engineering approach to synchronization protocols for compositional real-time systems.

REFERENCES

- [1] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symp.*, Dec. 2003, pp. 2–13.
- [2] T. Baker, "Stack-based scheduling of realtime processes," *Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, March 1991.
- [3] T. M. Ghazalie and T. P. Baker, "Aperiodic servers in a deadline scheduling environment," *Real-Time Syst.*, vol. 9, no. 1, pp. 31–67, 1995.
- [4] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Real-Time Systems Symp.*, Dec. 2006, pp. 257–267.
- [5] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Scheduling of semi-independent real-time components: Overrun methods and resource holding times," in *Conf. on Emerging Technologies and Factory Automation*, Sept. 2008, pp. 575–582.
- [6] G. Yao, G. Buttazzo, and M. Bertogna, "Comparative evaluation of limited preemptive methods," in *Conf. on Emerging Technologies and Factory Automation*, Sept. 2010.
- [7] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Syst.*, vol. 42, no. 1-3, pp. 63–119, Aug. 2009.
- [8] M. Behnam, T. Nolte, and R. J. Bril, "Tighter schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks," in *Conf. on Engineering of Complex Computer Systems*, April 2011.
- [9] M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Protocol-transparent resource sharing in hierarchically scheduled real-time systems," in *Conf. on Emerging Technologies and Factory Automation*, Sept. 2010.
- [10] Z. Deng and J.-S. Liu, "Scheduling real-time applications in open environment," in *Real-Time Systems Symp.*, Dec. 1997, pp. 308–319.
- [11] X. Feng and A. Mok, "A model of hierarchical real-time virtual resources," in *Real-Time Systems Symp.*, Dec. 2002, pp. 26–35.
- [12] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Real-Time Systems Symp.*, Dec. 1999, pp. 256–267.
- [13] G. Lipari and S. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *Proc. Real-Time Technology and Applications Symp.*, May 2000, pp. 166–175.
- [14] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Real-Time Systems Symp.*, Dec. 2007, pp. 129–138.
- [15] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems," in *Conf. on Embedded Software*, Oct. 2007, pp. 279–288.
- [16] M. Behnam, T. Nolte, M. Åsberg, and R. J. Bril, "Overrun and skipping in hierarchically scheduled real-time systems," in *Conf. on Embedded and Real-Time Computing Systems and Applications*, Aug. 2009, pp. 519–526.
- [17] I. Shin, M. Behnam, T. Nolte, and M. Nolin, "Synthesis of optimal interfaces for hierarchical scheduling with resources," in *Real-Time Systems Symp.*, Dec. 2008, pp. 209–220.
- [18] P. Gai, G. Lipari, and M. Di Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *Real-Time Systems Symp.*, Dec. 2001, pp. 73–83.
- [19] M. Bertogna, N. Fisher, and S. Baruah, "Static-priority scheduling and resource hold times," in *Parallel and Distributed Processing Symp.*, March 2007, pp. 1–8.
- [20] A. Burns, "Preemptive priority based scheduling: An appropriate engineering approach," in *Advances in Real-Time Systems*, S. Son, Ed. Prentice-Hall, 1994, pp. 225–248.
- [21] J. Regehr, "Scheduling tasks with mixed preemption relations for robustness to timing faults," in *Real-Time Systems Symp.*, Dec. 2002, pp. 315–326.
- [22] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 257–269, 2005.
- [23] Micrium, "RTOS and tools," March 2010. [Online]. Available: <http://micrium.com/>
- [24] J. J. Labrosse, *MicroC/OS-II*. R & D Books, 1998.
- [25] W. Cools, "Extending $\mu C/OS-II$ with FPDS and reservations," Master's thesis, Eindhoven University of Technology, July 2010.
- [26] R. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Real-Time Systems Symp.*, Dec. 2005, pp. 389–398.
- [27] OpenCores, (2009) OpenRISC overview. [Online]. Available: <http://www.opencores.org/project,or1k>