Timing Analysis for Resource Access Interference on Adaptive Resource Arbiters

Andreas Schranzhofer[♯], Rodolfo Pellizzoni[⋆], Jian-Jia Chen[‡], Lothar Thiele[♯], Marco Caccamo[†]
[♯] Swiss Federal Institute of Technology (ETH), Zurich, Switzerland {schranzhofer,thiele}@tik.ee.ethz.ch
[⋆] University of Waterloo, Waterloo, Ontario, Canada, rpellizz@uwaterloo.ca
[‡] Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany, jian-jia.chen@kit.edu
[†] University of Illinois at Urbana-Champaign, Urbana, IL, USA, mcaccamo@illinois.edu

***Abstract*—Modern multiprocessor and multicore architectures adopt shared resources to meet increased performance requirements. Adaptive arbiters, such as FlexRay, have been adopted to grant access to shared resources. While increasing the performance, timing analysis is more challenging with this kind of arbiter. This paper considers real-time tasks that are composed of superblocks, while superblocks themselves are composed of phases. Phases are characterized by their worst-case computation time on their processing element and their worst-case number of access requests to a shared resource. Resource accesses, such as access to caches or scratchpad memory, are synchronous and cause the processing element to stall until the access is served. Based on dynamic programming, we develop an algorithm that safely derives an upper-bound of the worst-case response time of a phase. The worst-case response time of a task can then be determined for both sequential or time-triggered execution of superblocks. Experimental results are conducted for a real-world application.**

***Keywords*—Worst-case timing analysis, shared resource access, real-time embedded systems, FlexRay protocol.**

I. Introduction

In the multicore era, the performance improvement w.r.t. computation depends on task parallelism, data parallelism, etc. In order to reduce hardware costs, commercial multicore platforms typically have shared resources, such as buses, main memory, and DMA. These shared resources are the new bottleneck for performance improvement.

Resource sharing in Commercial Off The Shelf (COTS) multicore systems is typically designed to improve average performance. However, when considering safety-critical embedded systems, such as controllers in Automotive Open System Architecture (AutoSAR) [1], bursty and irregular resource accesses to a shared resource might make the system miss its deadlines. Hence, interference due to contention on the shared resources has to be considered.

Multiprocessor scheduling has been studied in the literature for parallel systems without resource sharing, e.g., [5]. However, with shared resources, timing predictability becomes complicated, and has recently become an important topic both in the architectural design and the analytical tool design. Edwards and Lee [3] propose PRET machines to modify either memory arbitration or cache behavior for improving the timing predictability. Closely related to timing analysis for systems with shared resources, there are also works that focus on spatial interference or data conflicts caused due to cache accesses, e.g., Guan et al. [6] and Li et al. [8]. In [6], Guan et al. propose a scheduling algorithm and a schedulability test based on linear programming for multicore systems with shared L2 cache. Timing analysis, based on linear programming, considering cache conflicts is provided in [8].

Resource accesses might be asynchronous, e.g., message passing, or synchronous, e.g., memory accesses due to cache misses. For asynchronous resource accesses, the interference of the shared resource can be resolved by analyzing the required buffer size and the events that are produced to the resource. Approaches like Modular Performance Analysis [17], or SymTA/S [16] can be applied to analyze the worst-case response time. However, for synchronous accesses, an access request blocks the processing element until the access is served. As a result, the more the accesses to a shared resource of a task are blocked, the larger the worst-case response time (WCRT) will be.

To provide timing guarantees, several researchers, such as Pellizzoni et al. [9], [10] and Schliecker et al. [13], [14], have recently proposed methodologies to analyze the worst-case delay a task suffers due to accesses to a shared bus and shared memory, assuming synchronous resource accesses. Specifically, in [9], [10], a framework is developed to analyze the maximum delay that a task may suffer due to peripheral interference. Furthermore, a coscheduling algorithm is proposed to maximize the admitted peripheral traffic on the bus while the timing constraints of real-time tasks are satisfied. Schliecker et al. [13], [14] consider a system with a global shared resource, in which the maximum and the minimum numbers of accesses in particular time windows are assumed to be given. By assuming fixed-priority scheduling on the shared resource with statically assigned priorities to tasks, the worst-case interference is derived based on the access patterns of higher priority tasks.

The time division multiple access (TDMA) policy has been adopted in many industrial applications to improve timing predictability, to eliminate interference by task isolation, and to simplify schedulability analysis. By applying static analysis to compute the feasible traces, Rosen et al. [12] developed approaches to derive efficient TDMA arbitration policies. Schranzhofer et al. [15] developed a framework for analyzing the worst-case response time of real-time tasks when TDMA is applied to arbitrate access to a shared resource.

In order to be able to adapt to real-time tasks with early completion or less resource accesses, *adaptive resource ar-*

biters, such as the FlexRay communication protocol [4], have been proposed and adopted recently. An adaptive resource arbiter combines dynamic and static arbitration, in which an arbitration interval is divided into a static arbitration segment with a static slot to processing element assignment and a dynamic arbitration segment. As a result, it provides isolation between processing elements in the static arbitration segment, ensuring timing guarantees, and, in the dynamic segment, allows dynamic arbitration to improve the response time. Timing analysis of asynchronous resource accesses for the FlexRay communication protocol has been recently developed by Pop et al. [11] and by Chokshi et al. [2] whereas Lakshmanan et al. [7] consider hierarchical bus structures composed of FlexRay, CAN, etc. Nevertheless, synchronous resource access leads to blocking, which is not considered by these approaches.

For synchronous resource accesses with a TDMA resource arbiter, we can apply the results in [15], whereas the results in [10] can be adopted for a dynamic arbiter, such as the first-come-first-serve (FCFS) or the round-robin (RR) strategy. The adaptive resource arbiter requires the joint considerations of dynamic and static arbitration, and, hence, the analysis frameworks in [10], [15] cannot be directly applied. To the best of our knowledge, providing timing guarantees for synchronous resource accesses with an adaptive resource arbiter is an open problem. This paper considers systems with a shared resource, that requires a bounded amount of time to complete a request once access is granted. Access to the resource is synchronous and granted by an adaptive arbiter for at most one request at a time, resulting in blocking time for any other request. An ongoing access to a shared resource cannot be preempted. Moreover, we assume that the positions of accesses to the shared resource are not known a priori and neither is their order. We assume a hardware platform without timing anomalies, such as the fully timing compositional architecture proposed by Wilhelm et al. [18] and consider a given task partitioning, in which a task is allocated on a predefined processing element. Each task is divided into a set of superblocks, which are executed in a fixed sequence, and characterized by their worst-case number of accesses to the shared resource and their worst-case computation time. The contributions of this paper are as follows:

- Based on dynamic programming, we develop an algorithm that derives an upper-bound of the worst-case response time (WCRT) for superblocks and tasks, considering the delay caused by accesses to the shared resource.
- Our analysis generalizes the analysis in [10] for dynamic resource arbiters only and in [15] for static resource arbiters (TDMA) only.
- We present experimental results for a real-world application, and show that minor suboptimal decisions can result in significantly increased WCRTs.

In Section II, we introduce the proposed task model and model of the shared resource. We give an overview of our proposed algorithm in Section III and detail our notation and methodology in Section IV. Sections V and VI describe our WCRT analysis and proof its correctness. Experimental results are shown in Section VII. Section VIII concludes the paper.

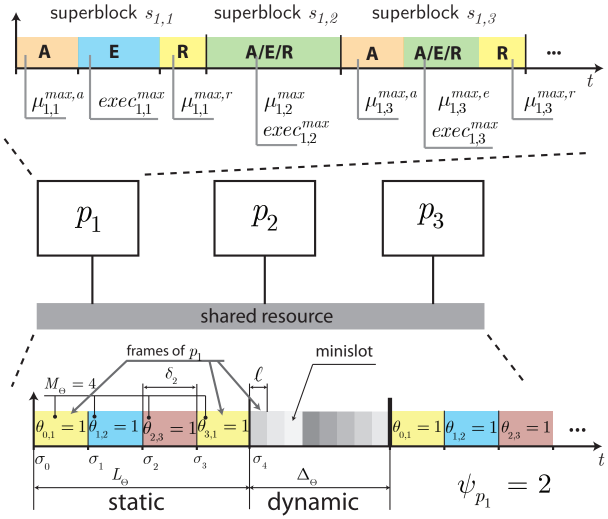

Fig. 1. An example for the adaptive arbiter.

II. System Model

This section presents the models of the tasks, the processing elements, the schedulers, and the resource arbiter of the shared resource.

A. Models of Tasks and Processing Elements

We consider the same models of tasks and processing elements as in [15], in which there is a set $\mathcal{P}$ of processing elements that execute independently. Processing elements share a common resource, for example, an interconnection fabric (bus) to access a shared memory. We assume tasks being constituted by a sequence of superblocks and compute a static schedule, such that a set of superblocks $\mathcal{S}_j$, comprised by a single or multiple tasks, executes periodically with period W_j on processing element $p_j \in \mathcal{P}$.

Superblocks are structured as (1) an *acquisition phase*, (2) an *execution phase*, and (3) a *replication phase*. The acquisition phase (the replication phase, respectively) of superblock $s_{i,j}$ is characterized by its maximum number of requests to access the shared resource $\mu_{i,j}^{max,a}$ ($\mu_{i,j}^{max,r}$, respectively); during such a phase, the superblock does not perform any relevant computation. The execution phase of superblock $s_{i,j}$ is characterized by its maximum number of requests to access the shared resource $\mu_{i,j}^{max,e}$ and its maximum required computation time $exec_{i,j}^{max}$ on processing element p_j. Superblock $s_{i,j}$ has a deadline $d_{i,j}$. As shown in Fig. 1, depending on the superblock code, it is possible for the acquisition/replication phase to be omitted or for the execution phase to contain no access requests.

We assume the parameters $\mu_{i,j}^{max,a}$, $\mu_{i,j}^{max,r}$, $\mu_{i,j}^{max,e}$, and $exec_{i,j}^{max}$ to be derived either by profiling and measurement or by applying static analysis for scratchpad memory to fetch new memory sections, etc. Note that $exec_{i,j}^{max}$ is derived by assuming that the resource accesses are done immediately, while $\mu_{i,j}^{max,a}$, $\mu_{i,j}^{max,r}$, and $\mu_{i,j}^{max,e}$ are derived by assuming that computation does not consume any time. For a superblock with logical branches, the above numbers might be overestimated, but the worst-case execution time is safely bounded.

$\mathcal{P}$	Set of processing elements	$s_{i,j}$	superblock i on processing element p_j	Θ	the arbiter
p_j	processing element j, $p_j \in \mathcal{P}$	$\rho_{i,j}$	starting time of superblock $s_{i,j}$	L_Θ	length of the static segment
W_j	processing cycle on p_j	$d_{i,j}$	rel. deadline of superblock $s_{i,j}$	Δ_Θ	length of the dynamic segment
C	completion time of an		w.r.t. the beginning of the task	σ_m	start time of slot m
	access request	$\mu_{i,j}^{max,[a\|e\|r]}$	number of access requests in	δ_m	length of slot m
M_Θ	Number of static slots		phases "a", "e" and "r"	$\theta(m, p_j)$	$\theta(m, p_j) = 1$ if slot m is assigned
ψ	number of static slots assigned to the	$exec_{i,j}^{max}$	computation time in $s_{i,j}$		to p_j and $\theta(m, p_j) = 0$ otherwise
	processing element under analysis	$\mathcal{S}_j$	sequence of superblocks executing on p_j	ℓ	the length of a minislot

TABLE I
SYMBOLS FOR THE SYSTEM MODEL

For a detailed discussion about the above models, please refer to [15]. Tab. I gives a summary of symbols used throughout this paper.

B. Model of the Shared Resource

Following the model in [15], the shared resource is only able to serve at most one access request at any time; a resource arbiter decides which request is granted. In other words, resource accesses are assumed *non-buffered*, for example, shared memory access due to cache misses or scratchpad memory misses, and results in the application to wait until an issued access is served. In this paper, we consider an adaptive arbiter, that follows the FlexRay protocol, where arbitration is conducted based on a sequence of *arbitration rounds*. Each round comprises a *static segment* followed by a *dynamic segment*. Once access to the shared resource is granted, the arbiter, denoted Θ, serves an access request within C units of time. We assume that processing elements and the resource arbiter initialize synchronously, such that the first arbitration round of Θ and the first superblock on each processing element start at time 0.

The static segment comprises a sequence of M_Θ (time) slots, indexed from 0 to $M_\Theta - 1$. Let L_Θ be the length of the static segment, while σ_m is the starting time of slot m relative to the beginning of the round, with $\sigma_0 = 0$. For ease of notation, we define $\sigma_{M_\Theta} = L_\Theta$; the duration of slot m is then $\delta_m = \sigma_{m+1} - \sigma_m$. Each slot serves requests of a single processing element according to function $\theta(m, p_j)$, i.e., $\theta(m, p_j) = 1$ if slot m is assigned to p_j and $\theta(m, p_j) = 0$ otherwise. Multiple slots can be assigned to the same processing element. A request of p_j in its assigned slot m is served only if it can be completed within the slot, i.e., it must arrive at least C time units before the end of the slot.

The dynamic segment of arbiter Θ is defined by its length Δ_Θ and the length ℓ of a *mini slot*. All processing elements contend for access to the shared resource during the dynamic segment, and again a request is granted only if it can be completed within the dynamic segment. Two arbitration models are possible. Under *constrained* request arbitration, resource accesses are granted in FCFS order at the beginning of each minislot, while *greedy* arbitration grants resource accesses in FCFS order, independent of their arrival time. The FlexRay communication protocol is a special case of the above definition when the dynamic arbitration segment is constrained and static slots all have the same duration. An example is shown in Fig. 1. Due to space limitation, we will focus our presentation for constrained request arbitration, i.e., FlexRay communication protocol. The results can be easily applied for greedy request arbitration and RR arbitration for the dynamic arbitration segment with minor changes, which are similar to the extension from FCFS to RR in [10]. We assume $\Delta_\Theta + L_\Theta$ to be an integer multiple of the period W_j of the set of superblocks S_j under analysis.

C. Problem Definition

In this paper we study the problem of deriving the worst-case response time (WCRT) of the tasks executing on a processing element $p_j \in \mathcal{P}$. Accesses to the shared resource are synchronous, i.e., once an access request is issued by a task/superblock, it has to stall until this access request is served by the shared resource. We assume an adaptive arbiter Θ, following the constrained request arbitration, granting access to the shared resource. The length of an arbitration round $\Delta_\Theta + L_\Theta$ and the period W_j are integer multiples. Previous work considered dynamic arbitration [10] or static arbitration [15] only. Thus, these approaches are incapable of considering the interdependency of the delays caused by slotted arbitration in the static segment and the delay caused by contention in the dynamic segment. This paper proposes a generalization of these studies to allow the analysis of adaptive resource arbiters. An adaptive schedule for the shared resource is said to be *schedulable* if all the superblocks on all processing elements can finish before their respective deadlines.

III. ANALYSIS OVERVIEW

In our analysis we propose an algorithm that derives the worst-case execution trace for the sequence of superblocks $\mathcal{S}_j$ that executes on element p_j. In other words, we derive the trace that results in the WCRT for the task. The degree of freedom for constructing this worst-case trace is the sequence of access requests and the computation that have to be performed in the individual superblocks. Delays due to shared resource contention come from two cases: (1) an access request happens during the static segment, but the current slot is assigned to another processing element or there is not enough time to complete the request; or (2) an access request happens during the dynamic segment of the arbiter, but other access requests are already in the queue or there is not enough time to complete the request. In both cases, the access request has to wait either its turn in the dynamic segment or until the beginning of its next assigned static slot, whichever comes first. To simplify the discussion, we will refer to time slots assigned to processing element p_j as "frames of p_j". The dynamic segment is a frame for every processing element p_j as well, since all elements contend for access to the shared

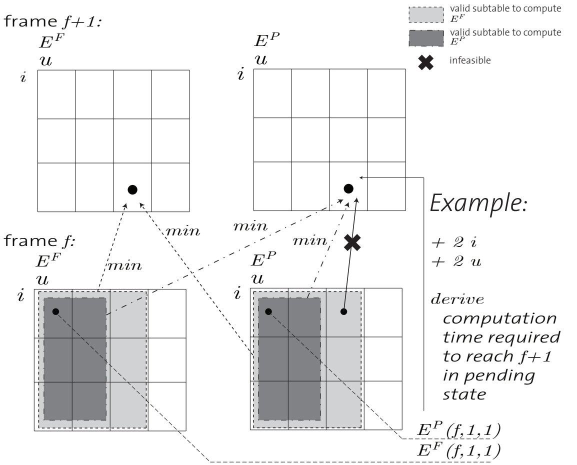

Fig. 2. Example how to construct the dynamic programming table.

resource during the dynamic segment. We propose a dynamic programming algorithm to solve this problem iteratively.

The key idea is as follows. Let u be the number of access requests performed by the sequence $\mathcal{S}_j$, and i be the number of requests performed by tasks running on other cores that *interfere* (delay through contention) with $\mathcal{S}_j$ during dynamic segments. Our algorithm derives the *minimum* amount of time that the superblocks in $\mathcal{S}_j$ must compute to reach the beginning of a particular frame f, for any feasible value of u and i; note that increasing u and/or i can increase the resource access delay suffered by the task, hence less computation time is required to reach a given frame. Based on this information, the algorithm then iteratively derives the minimum amount of computation time required to reach the following frame $f+1$, and so on and so forth until the required computation time to reach frame $f'+1$ becomes greater than the worst-case computation time of the task for any value of u and i; this implies that in the worst case the task can only reach up to frame f'.

Due to the blocking behavior of access requests, a frame f can be reached in the "free" state, i.e., the processing element that executes $\mathcal{S}_j$ has no unserved access requests upon activation of frame f, or in the "pending" state, i.e., the processing element stalls, because there is an unserved access request that has to be served immediately upon activation of frame f. The minimum amount of computation time is then stored in two tables $E^F(f,u,i)$ and $E^P(f,u,i)$ for the free and pending states, respectively.

Consider Fig. 2 as an example to compute the minimal computation time. We want to determine the minimum amount of computation time $E^F(f+1,u,i)$ and $E^P(f+1,u,i)$ that is required to reach frame $f+1$ in the free and pending state, respectively, for a particular combination of u access requests and i interferences. Then, this value depends on the values computed for the previous frame, $E^F(f,u',i')$ and $E^P(f,u',i')$, for all values of u', i' that are compatible with u, i: note that $u-u'$ and $i-i'$ represent the amount of performed access requests and suffered interferences, respectively, between the beginning of frames f and $f+1$. Clearly, it must hold $i' \leq i$. Furthermore, u' must be strictly smaller than u to be able to reach frame $f+1$ in the pending state, since at least one access request needs to be issued after frame f is reached to enter the pending state. This condition is not required if frame $f+1$ is reached in the free state. In other words, computing value $E^P(f+1,u,i)$ depends on the values $E^F(f,u',i')$ and $E^P(f,u',i')$ for the previous frame, for all combinations of $u < 3$ and $i \leq 3$ (dark gray field in Fig. 2); while $E^F(f+1,u,i)$ depends on the values $E^F(f,u',i')$ and $E^P(f,u',i')$ for $u \leq 3$ and $i \leq 3$ (light gray field in Fig. 2). In the remainder of this paper, we will detail how to derive the amount of computation time that is required to move from one configuration to the other, i.e., how to compute the minimum amount of computation time that is required to get from $E^F(f,u',i')$ or $E^P(f,u',i')$ to $E^F(f+1,u,i)$ or $E^P(f+1,u,i)$. Furthermore, we will show how to initialize the tables for the first slot and how to compute the final WCRT when the iteration stops.

IV. Analysis Methodology

To simplify the derivation of WCRT bounds, we split our analysis in two parts. Algorithm 1, described in Section V, computes the WCRT of a single phase, based on its maximum amount of access requests $\mu_{i,j}^{max,[a|e|r]}$ to the shared resource and its maximum amount of computation time $exec_{i,j}^{max}$. Algorithm 2, which will be detailed in Section VI, is then used to compute the WCRT of a complete superblock and task. The start time of a phase t^s equals the completion time of the preceding phase, in case of sequential execution, or a specified starting time, in case of time-triggered execution. The first phase of the first superblock starts at time 0. Depending on the start time of a phase, the *initialization* stage of Algorithm 1 (lines 2-7) initializes the tables for the dynamic programming approach. The second stage, or the *analysis* stage (lines 8-21), performs the dynamic programming analysis described in Section III. This stage iterates until the minimum amount of computation required to reach frame $f'+1$ exceeds the maximum amount of computation $exec_{i,j}^{max}$. The *finalization* stage (lines 22-23) then derives the WCRT of the phase based on f' and the dynamic programming tables E^F and E^P.

Notations used in the analysis for a phase: To simplify notations, we drop the subscripts i,j from superblock $s_{i,j}$ and consider a phase under analysis defined by parameters $\{t^s, \mu^{\max}, exec^{\max}\}$. For an acquisition or replication phase, we simply set $exec^{\max} = 0$, and $\mu^{\max} = \mu_{i,j}^{\max,a}$, or $\mu^{\max} = \mu_{i,j}^{\max,r}$, respectively; while for an execution phase, we set $exec^{\max} = exec_{i,j}^{\max}, \mu^{\max} = \mu_{i,j}^{\max,e}$.

Consider p_j to be the processing element on which the phase under analysis executes. Then we define ψ to be the total number of static slots assigned to processing element p_j, i.e., $\psi = \sum_{0 \leq m < M_\Theta} \theta(m,p_j)$. Since both assigned static slots and the dynamic segment count as frames of p_j, it follows that p_j has $\psi+1$ frames in each arbitration round. To simplify the algorithm description, we introduce some notations describing important properties of each frame. We use I to represent the ordered set of the indexes of frames assigned to p_j. Static slots have indexes from 0 to $M_\Theta - 1$ as detailed in Section II-B, while we assign index M_Θ to the dynamic segment, such that $I = \{I_0, \ldots I_i, \ldots I_\psi\}$, where $I_\psi = M_\Theta$ represents the

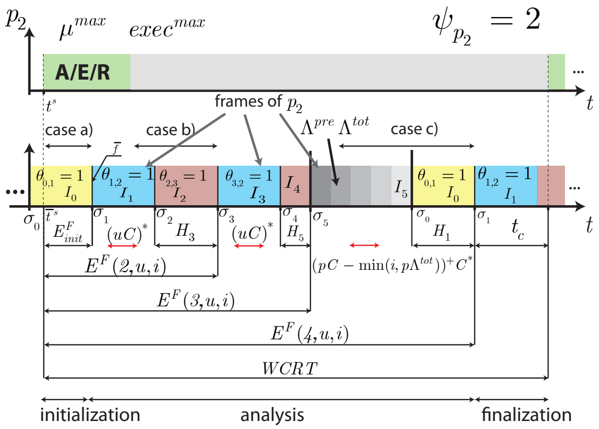

Fig. 3. Analysis Overview

dynamic segment. As an example, consider Fig. 3, where static slots 1 and 3 are assigned to processing element p_2, and the dynamic segment has index 5. Then the frame indexes of p_2 are $I = \{I_0 = 1, I_1 = 3, I_2 = 5\}$. Finally, let H_{I_f} (with $0 \leq f \leq \psi$) be the time distance between the end of frame I_f and the beginning of the following frame. Note that the end of the dynamic segment (frame ψ) corresponds to the end of the arbitration round. Then H_{I_f} can be written as:

$$H_{I_f} = \begin{cases} \sigma_{I_{f+1}} - (\sigma_{I_f} + \delta_{I_f}) & \text{for } 0 \leq f < \psi, \\ \sigma_{I_1} & \textit{otherwise}. \end{cases} \quad (1)$$

Our proposed algorithm computes the WCRT of a phase, by finding frame f' that can be reached by the worst-case trace. Depending on the phase under analysis, many arbitration rounds might be necessary until this frame can be found. Therefore, the amount of arbitration rounds that have been performed needs to be considered. Let r_0 be the arbitration round, during which the phase is activated, i.e., the w-th round such that $w \cdot (L_\Theta + \Delta_\Theta) \leq t^s < (w+1) \cdot (L_\Theta + \Delta_\Theta)$, and r_i be the $(w+i)$-th round. Then frame 0 is the first frame of p_j in round r_0 and frame f is frame $(f \mod (\psi+1))$ of p_j in round $r_{\lfloor f/(\psi+1) \rfloor}$. For notational simplicity, we set $H_{I_f} = H_{I_{f \mod (\psi+1)}}$; analogously, we translate the starting time t^s of a phase to the starting time $\bar{t}^s$ relative to the beginning of the current round.

Interference during the dynamic segment of arbiter Θ depends on the access pattern of the other processing elements to the shared resource. We capture this pattern through a set of *arrival curves*: arrival curve $\alpha_k(\Delta)$ represents the maximum amount of resource access requests that can be produced by tasks running on processing element p_k during a time window of length Δ. Pellizzoni et al. [10] introduced a methodology to derive arrival curves for the different superblock models (sequential or time-triggered) that we consider in this work. We translate the $\alpha_k(\Delta)$ representation, such that $\alpha_k(f)$ represents the maximum amount of resource accesses by p_k in a time window of length equal to the interval between the start time of the phase under analysis and the beginning of frame f. Consider multiple interfering processing element for a phase executing on processing element p_j, e.g., a set of interferers IP, such that $IP \subseteq \mathcal{P} \setminus p_j$. Then we approximate the overall interference of these interferers to the phase under analysis as the sum of the individual interferences, i.e., $\alpha(f) = \sum_{p_k \in IP} \alpha_k(f)$. This is a pessimistic assumption, but to the best of our knowledge, deriving tight bounds on the combined interference is still an open problem. In fact, obtaining an exact interference pattern, for multiple competing processing elements, onto the superblock/phase under analysis is exponential in the number of interfering phases [10], which makes it intractable in most practical settings. Finally, interference is affected by two more parameters: (1) the number of processing elements Λ^{tot} that can interfere with the phase under analysis and (2) the number of processing element Λ^{pre}, that can have pending access requests that are serviced before a pending request of p_j at the beginning of the dynamic segment, i.e., the number of interfering access requests once the dynamic segment becomes active. As we show in Section V, the worst-case pattern in this situation is produced when the phase under analysis becomes pending in slot $I_{\psi-1}$, its last assigned slot in the static segment. Since arbitration in the dynamic segment is performed in FCFS order, Λ^{pre} includes all processing element that can interfere with the phase under analysis and are assigned at least one static slot with index smaller than $I_{\psi-1}$. Given a tight interference representation, our proposed algorithm performs a tight worst-case response time (WCRT) analysis.

Algorithm 1 Analyze a single phase

```
procedure AnalyzePhase($t^s$, $exec^{max}$, $\mu^{max}$, $\Theta$, $IP$)
    $\forall f, u, i : E^P(f,u,i) := +\infty, E^F(f,u,i) := +\infty$
    $\bar{t}^s = t^s - \lfloor \frac{t^s}{L_\Theta + \Delta_\Theta} \rfloor \cdot (L_\Theta + \Delta_\Theta)$
    $\bar{f} = \min\left( (\psi+1), \{f | 0 \leq f \leq \psi \wedge \bar{t}^s \leq \sigma_{I_f}\} \right)$
    $\forall u, i$ : compute $E^F(\bar{f}, u, i), E^P(\bar{f}, u, i)$ by Eq. 2 and Eq. 3 resp.
    $reachable = true$
    $f = \bar{f} - 1$
    while $reachable$ do
        $reachable = false$
        $f = f + 1$
        for $\forall u \in \{0 \dots \mu_{max}\}$ and $i \in \{0 \dots \alpha(f+1)\}$ do
            if $f \mod \psi + 1 < \psi$ then
                compute $E^P(f+1,u,i), E^F(f+1,u,i)$ by Eq. (4), (6)
            else
                compute $E^P(f+1,u,i), E^F(f+1,u,i)$ by Eq. (5), (7)
            end if
            if $\min(E^P(f+1,u,i), E^F(f+1,u,i)) \leq exec^{\max}$ then
                $reachable = true$
            end if
        end for
    end while
    compute $t_c$ by Eq. 8 or 9 for $f$ as static or dynamic frame resp.
    return $WCRT = \left\lfloor \frac{f}{\psi+1} \right\rfloor \cdot (L_\Theta + \Delta_\Theta) + \sigma_{I_f} + t_c - t^s$
end procedure
```

V. Analysis for a Single Phase

In this section, we will introduce the three stages required to derive the WCRT for a single phase. The two data structures $E^F(f,u,i)$ and $E^P(f,u,i)$ represent the minimum amount of time required to reach a particular frame f such that u access requests have been served and i interferences have been suffered, for $u \in \{0 \dots \mu_{max}\}$ and $i \in \{0 \dots \alpha(f)\}$. Section V-A details on the initialization phase, which depends on the starting time t^s of the phase under analysis. Section V-B introduces the analysis of a single phase, based on a dynamic programming approach. The algorithm iterates until a frame f' is reached, such that each entry in $E^F(f'+1,u,i)$ and $E^P(f'+1,u,i)$, $\forall u,i$, exceeds $exec^{max}$. Section V-C shows how to derive the final WCRT for the phase under analysis, based on the previously computed data structures E^F and E^P.

A. Initialization Stage

The initialization stage computes the amount of computation that is required to reach the first frame $\bar{f}$ after the activation of a phase at time t^s. Depending on this starting time, the phase is activated either (a) within the static segment of arbiter Θ but the slot is not assigned to p_j or (b) within the static segment and within a frame (i.e., a slot assigned to p_j) or (c) within the dynamic segment - compare cases "a)", "b)" and "c)" in Fig. 3. Note that time windows that depend on variable parameters in Fig. 3 are marked with an asterisk (*) as superscript - namely time windows that depend on u, i and Λ^{tot}.

First, in "case a)", the amount of time required to reach frame $\bar{f}$ is only related to the amount of time between the activation time t^s and the start time of the next frame. Frame $\bar{f}$ can be reached in the pending state, if an access request can be issued immediately upon the activation of the phase, i.e., no computation is performed and $E^P(\bar{f},1,i)=0,\forall i$, see Eq. (3), case 2.[1] If no access request can be issued, frame $\bar{f}$ cannot be reached in the pending state, i.e., $E^P(\bar{f},1,i)=\infty,\forall i$, see Eq. (3), case 1. Frame $\bar{f}$ can be reached in the free state by performing computation from the activation of the phase until the activation of frame $\bar{f}$, hence $E^F(\bar{f},u,i)=\sigma_{I_{\bar{f}}}-\bar{t}^s,\forall u,i$, see Eq. (2), case 1.

Second, "case b)", considers the case, that the current slot is a frame of p_j, i.e., the phase can immediately issue an access request that will be granted by the arbiter. Therefore, $E^F(\bar{f},u,i)$ is a function of the current frames remaining duration, the amount of issued access requests u and the time between the current frame $\bar{f}-1$ and the next frame $\bar{f}$, see Eq. (2), second case. $E^P(\bar{f},u,i)$ is computed analogously, except that the last access request is issued $C-\epsilon$ time units before the frame expires.[2] As a result, this request is not served anymore, the phase has to stall until the activation of frame $\bar{f}$ and no computation can be performed between the current and the next slot, see Eq. (3) - third case.

Third, "case c)", considers the phase to start in the dynamic slot. For both cases, reaching slot $\bar{f}$ in the pending or in the free state, the required amount of computation is related to the distance of the dynamic frames activation time, the amount of access requests and the amount of interference (denoted as $\min(i,u\Lambda^{tot})C$ - will be explained in more detail in Section V-B). In order to reach frame $\bar{f}$ in the pending state, computation is performed, such that the last request is issued $C-\ell$ time units before the expiration of the dynamic segment. As a result, the request cannot be served in the dynamic segment anymore and the processing element has to stall until the activation of frame $\bar{f}$, i.e., the element is in pending state upon activation of frame $\bar{f}$, see Eq. (3) - fourth case. In contrast to that, frame $\bar{f}$ is reached in the free state, if computation is performed during the current frames expiration and the next frames activation, denoted $H_{I_{\bar{f}-1}}$, see Eq. (2) - third case.

Algorithm 1 shows the derivation of the WCRT of a single phase. The input to the algorithm is the activation time t^s, the maximum amount of computation $exec^{max}$, the maximum amount of access requests μ^{max}, the arbiter Θ and the set of interferers IP for the phase under analysis. First the data structures E^P and E^F are initialized to ∞. Then the starting time relative to the current arbitration round $\bar{t}^s$ and the next frame $\bar{f}$ for the phase under analysis are computed, see Line (3) and (4). Data structures $E^P(\bar{f},u,i), E^F(\bar{f},u,i)$ are initialized for all u and i, as described in this section, see Line (5) and the termination condition *reachable* is set to *true*.

B. Analysis Stage

After the initialization is done, the main loop of Algorithm 1 starts in Line (8). This loop computes the minimum amount of computation time that is required to reach the next frame for all combinations of u served access request, i suffered interferences for states "free" and "pending", see Line (13) in case the frame f is a static slot and Line (15) in case frame f is the dynamic segment.

The loop iterates until there is no entry in neither E^P nor E^F, that exceeds the maximum amount of computation time $exec^{max}$ for the phase under analysis. In other words, there exists no trace such that frame $f+1$ can be reached. Then *reachable* is not set to true, and the loop terminates. As a result, the last iteration of the algorithm computes values E^P and E^F for the unreachable frame $f+1$. See Fig. 1 for an example to compute $E^F(4,u,i)$, assuming $E^F(5,u,i)$ cannot be reached for any combination of u and i. Therefore, the finalization phase computes the final WCRT based on the previous frame, i.e., frame f, since the WCRT of the worst-case trace must be between frames f and $f+1$.

Lemma 1: Algorithm 1 terminates, for phases with finite values for parameters $exec^{max}$ and μ^{max}, and an arbiter with frame sizes larger than or equal to C.

Proof: Every iteration of the Algorithm increments the variable f, that denotes the next frame. For finite parameters $exec^{max}$ and μ^{max}, and an arbitration policy that guarantees at least one access request to be served per cycle, the amount of frames (and arbitration cycles) that is required to finish this phase equals to at most the sum of the amount of access requests μ^{max} and the maximum amount of computation $exec^{max}$ that have to be performed. In order to reach the beginning of frame $f+1$ from frame f, at least one access request has to be issued $C-\epsilon$ time units before f expires, i.e., $f+1$ is reached in the pending state. Otherwise, computation amounting for H_{I_f} is performed between frame f and $f+1$, i.e., $f+1$ is reached in the free state.

Therefore, there exists a $f \leq \mu^{max} + \left\lfloor \frac{exec^{max}}{\min\limits_{\forall f} \delta_{I_f} + H_{I_f}} \right\rfloor$, such that *reachable* is not set to *true* and the Algorithm 1 terminates. ■

[1] If $z>0$, function z^+ is z; otherwise z^+ is 0

[2] ϵ is an arbitrary small value, greater than 0

$$E^F_{init}(\bar{f},u,i) = \begin{cases} \sigma_{I_{\bar{f}}} - \bar{t}^s & \bar{t}^s \leq \sigma_{I_0} \vee (\bar{f} > 0 \wedge \bar{t}^s \geq \sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}}) \\ (\sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}} - \bar{t}^s - u \cdot C)^+ + H_{I_{\bar{f}-1}} & \bar{f} < \psi + 1 \\ (\sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}} - \bar{t}^s - u \cdot C - \min(i, u \cdot \Lambda^{tot}) \cdot C)^+ + H_{I_{\bar{f}-1}} & \text{otherwise} \end{cases} \tag{2}$$

$$E^P_{init}(\bar{f},u,i) = \begin{cases} +\infty & u = 0 \\ 0 & \bar{t}^s \leq \sigma_{I_0} \vee (\bar{f} > 0 \wedge \bar{t}^s \geq \sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}}) \\ (\sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}} - \bar{t}^s + \epsilon - u \cdot C)^+ & \bar{f} < \psi + 1 \\ (\sigma_{I_{\bar{f}-1}} + \delta_{I_{\bar{f}-1}} - \bar{t}^s + \ell - u \cdot C - \min(i, u \cdot \Lambda^{tot}) \cdot C)^+ & \text{otherwise} \end{cases} \tag{3}$$

$$E^F(f+1,u,i) = \min_{0 \leq p \leq u} \begin{cases} E^F(f,u-p,i) + (\delta_{I_f} - pC)^+ + H_{I_f} \\ E^P(f,u-p,i) + (\delta_{I_f} - (p+1)C)^+ + H_{I_f} \end{cases} \tag{4}$$

$$E^F(f+1,u,i) = \min_{\substack{0 \leq p \leq u \\ 0 \leq l \leq i}} \begin{cases} E^F(f,u-p,i-l) + (\Delta_\Theta - pC - \min(l, p\Lambda^{tot})C)^+ + H_{I_f} \\ E^P(f,u-p,i-l) + (\Delta_\Theta - (p+1)C - \min(l, p\Lambda^{tot} + \Lambda^{pre})C)^+ + H_{I_f} \end{cases} \tag{5}$$

$$E^P(f+1,u,i) = \min_{1 \leq p \leq u} \begin{cases} E^F(f,u-p,i) + (\delta_{I_f} + \epsilon - pC)^+ \\ E^P(f,u-p,i) + (\delta_{I_f} + \epsilon - (p+1)C)^+ \end{cases} \tag{6}$$

$$E^P(f+1,u,i) = \min_{\substack{1 \leq p \leq u \\ 0 \leq l \leq i}} \begin{cases} E^F(f,u-p,i-l) + (\Delta_\Theta + \ell - pC - \min(l, p\Lambda^{tot})C)^+ \\ E^P(f,u-p,i-l) + (\Delta_\Theta + \ell - (p+1)C - \min(l, p\Lambda^{tot} + \Lambda^{pre})C)^+ \end{cases} \tag{7}$$

E^P and E^F are computed differently, depending on whether the next frame $f+1$ is a static slot of the dynamic segment. The following two sections will detail on the differences.

1) static slot: The minimum amount of computation that is required to reach frame $f+1$, if frame f is a static frame, is computed in Eq. (4) and (6), for the free and the pending state respectively. Eq. (4) computes the minimum amount of computation that is required to reach frame $f+1$, by considering $E^F(f, u-p, i)$ and $E^P(f, u-p, i)$, the duration δ_{I_f} of frame f, the time spent serving p access requests, $\forall 0 \leq p \leq u$, and the time between frames f and $f+1$, denoted H_{I_f}. As a result, $E^F(f+1, u, i)$ computes as the minimum amount of time that is required to reach frame $f+1$, considering all combinations of $u-p$ access requests served previous to frame f and p access requests served in frame f, for the "free" as well as for the "pending" state. Frame $f+1$ is reached in the free state, and therefore Eq. (4) considers computation to be performed in between frame f and $f+1$, i.e., by the term H_{I_f}.

Analogously, Eq. (6) computes how to reach frame $f+1$ in the pending state. An access request is issued $C-\epsilon$ units of time before frame f expires, resulting in this request not being served during frame f anymore. This way, the stall time of the element is maximized, and thus the amount of computation that is performed during frame f (at most $\delta_{I_f} + \epsilon$) and between frame f and $f+1$ (equals 0 since H_{I_f} is neglected) is minimized. Issuing the access request at any later point results in an increased amount of computation that has to be performed, and thus not in the worst-case trace.

Note that in Eq. (4) and (6), the amount of served access requests during frame f is increased by 1 for the pending states. This is due to the fact that upon the activation of frame f, an access request will be issued immediately. Furthermore, note that the variable p for Eq. (6) starts at 1, since at least a single access request must remain unserved, otherwise frame $f+1$ cannot be reached in the pending state.

Lemma 2: If frame f is a static slot, Eq. (4) and Eq. (6) compute the minimum amount of computation that have to be performed, such that frame $f+1$ can be reached in the free and pending state, respectively, for u served access requests and i interfering access requests by other phases.

Proof: For a particular frame $f+1$ and number of served access requests u, Eq. (4) and Eq. (6) compute the required amount of computation for all possible values of already served access requests $u-p$ (before frame f) and access requests p served in frame f, based on the pending and the free state. The minimum among these values, based on both the pending and the free state, is returned, and therefore Eq. (4) and (6) result in the minimum amount of computation time that is required to reach frame $f+1$ in the free state and pending state respectively. ■

2) dynamic slot: In case frame f is a dynamic slot, interfering access requests have to be considered. The minimum amount of computation that is required to reach frame $f+1$, is computed in Eq. (5) and (7), for the free and the pending state respectively. The respective amount of computation is derived for all possible remaining access requests p and remaining interfering access requests l. The minimum of these values is considered for further computation and stored in E^P and E^F, see Lemma 3.

Eq. (5) computes the amount of computation that is required to reach frame $f+1$ in the free state by considering the amount of computation required to reach the current frame f, the length of the dynamic segment Δ_Θ, the amount of served access requests p (or $p+1$ for the pending case), the amount of access requests that can be interfered with, and the time between the expiration of the current and the activation of the next frame H_{I_f}. The term $\min(l, p\Lambda^{tot})C)^+$ in Eq. (5), for the free state, constrains the maximum amount of interference a phase can suffer during frame f. Intuitively, $p\Lambda^{tot}$ represents the property that each access request issued in frame f can be interfered by every other processing element once, i.e., each access request ends up in the last position of the FIFO queue of the arbiter during the dynamic segment. On the other hand, in case there is less interference, i.e., l, then the access requests issued during frame f cannot suffer more interference. For E^F based on the pending state, this term changes to $\min(l, p\Lambda^{tot} + \Lambda^{pre})C)^+$. In other words, upon activation of frame f there is a pending access request that has to be served and it is interfered by up to Λ^{pre} access requests.

Eq. (7) computes the amount of computation that is required to reach frame $f+1$ in the pending state, similarly to Eq. (5). However, frame $f+1$ shall be reached in the pending state, and therefore, the term H_{I_f} is not present in Eq. (7). Instead, the amount of computation during the active frame is increased, such that the last access request cannot be served anymore. That is, in comparison to Eq. (5), the amount of computation is increased by one minislot of size ℓ. The last access request cannot be served anymore and the phase stalls until frame $f+1$ is activated.

Lemma 3: If frame f is a dynamic slot, Eq. (5) and (7) compute the minimum amount of computation that have to be performed, such that frame $f+1$ can be reached in the free and pending state respectively, for u served access requests and i interfering access requests by other phases.

Proof: Assume frame $f+1$ should be reached, with u access requests being served and i interfering access requests being issued by other processing elements. Then Eq. (5) and (7) compute the amount of computation that is required to reach frame $f+1$ for all possible combinations of access request, that are already served at frame f, denoted $u-p$, and access requests that have to be served during frame f, denoted p. Similarly, the interferences that have already happened, denoted $i-l$, and those that can happen during frame f, denoted l, are considered. The rest of the proof is similar to the proof for Lemma 2. ■

C. Finalization

Algorithm 1 iterates until frame $f+1$, that cannot be reached anymore, is found, i.e., frame $f+1$ such that variable *reachable* remains *false* for any u, i, see Line 17. Then the worst-case completion of the phase under analysis is between frame f and $f+1$. As a result, the data structures $E^F(f,u,i)$ and $E^P(f,u,i)$, $\forall u,i$ are considered to compute the final WCRT.

Eq. (8) and Eq. (9) compute the WCRT of a phase, in case frame f is a static or a dynamic frame, respectively.

$$t_c = \max_{\forall u,i} \begin{cases} (exec^{max} - E^F(f,u,i) + (\mu^{max} - u)C)^+ \\ (exec^{max} - E^P(f,u,i) + (\mu^{max} - u + 1)C)^+ \end{cases} \tag{8}$$

$$t_c = \max_{\forall u,i} \begin{cases} (exec^{max} - E^F(f,u,i) + (\mu^{max} - u + \\ \qquad \min(\alpha(f+1) - i, (\mu^{max} - u)\Lambda^{tot}))C)^+ \\ (exec^{max} - E^P(f,u,i) + (\mu^{max} - u + 1 + \\ \qquad \min(\alpha(f+1) - i, (\mu^{max} - u)\Lambda^{tot} + \Lambda^{pre}))C)^+ \end{cases} \tag{9}$$

Consider frame f to be a static frame, then Eq. (8) computes the response time t_c of a phase by considering the remaining access requests $(\mu^{max} - u)$ and the phases computation time $(exec^{max})$. Their difference represents the remaining amount of computation that has to be performed, but proved to be insufficient to reach frame $f+1$.

Similarly, Eq. (9) computes the response time t_c of a phase, if frame f is the dynamic segment. In this case, the interference that might be suffered during that frame has to be considered. As in Eq. (5) and (7), the additional delay due to inference depends on the upper bound of interference $(\alpha(f+1) - i)$, and the number of issued access request that can be interfered with: $(\mu^{max} - u)\Lambda^{tot}$ in the free state, and $(\mu^{max} - u)\Lambda^{tot} + \Lambda^{pre}$ in the pending state. The WCRT t_c is the maximum resulting value for any u and i.

Lemma 4: Given a constrained adaptive arbiter Θ, Algorithm 1 computes an upper bound of the worst-case response time for a phase with a defined starting time t^s, amount of access requests μ_{max}, computation time $exec^{max}$, and the set of interferers IP.

Proof: $E^F(f+1,u,i)$ and $E^P(f+1,u,i)$ are computed as the minimum of $E^F(f, u-p, i-l)$ and $E^P(f, u-p, i-l)$, for p and l as defined in Eq. (4) to (7). Instead of considering the minimum value for $E^F(f+1,u,i)$ and $E^P(f+1,u,i)$, consider all values that are computed in Eq. (4) to (7). In other words, at each iteration, all possible distributions of (a) already served access requests $u-p$ and remaining access requests u, and (b) suffered interferences $i-l$ and interferences that can be suffered in the current frame i, and their respective required computation time are stored in a data structure, not only their minimum. Consider $E^F_n(f+1,u,i)$ and $E^P_n(f+1,u,i)$ to be the amount of computation that is required to reach frame $f+1$, for any distribution of u and i, but the one that results as the minimum. In other words, $\forall n : E^F_n(f+1,u,i) > E^F(f+1,u,i)$ and $\forall n : E^P_n(f+1,u,i) > E^P(f+1,u,i)$. Then the amount of computation that remains after frame $f+1$ is reduced, or $exec^{max} - E^F_n(f+1,u,i) < exec^{max} - E^F(f+1,u,i)$ and $exec^{max} - E^P_n(f+1,u,i) < exec^{max} - E^P(f+1,u,i)$. Since $E^F_n(f+1,u,i)$ computes as the sum of $E^F_n(f,u,i)$ and a term related to the arbitration in frame f, $E^F_n(f+1,u,i) > E^F(f+1,u,i)$. Similarly, $E^P_n(f+1,u,i) > E^P(f+1,u,i)$.

Eventually the remaining computation time is insufficient to reach the next frame $f+1$ and the final response time is computed in Eq. (8) and (9). Since $E^F_n(f,u,i) > E^F(f,u,i)$ and $E^P_n(f,u,i) > E^P(f,u,i)$, the resulting response time t_c is reduced and so is the WCRT.

Conclusively, considering any other amount of computation to reach frame $f+1$, than the minimum computed in Eq. (4) to (7), leads to a decreased response time, which is upper-bounded by the solution derived from Algorithm 1. ■

D. Complexity

The complexity of the proposed methodology depends on the number of access requests and interference. Consider Algorithm 1, Line 15, then it is clear, that Eq. (5) and (7) are computed f times. The complexity of computing Eq. (5) and Eq. (7) is $O((\mu^{max})^2\alpha(f)^2)$, since for every combination of u and i, the minimum amount of computation has to be computed for all possible remaining access requests p and interferences l.

Furthermore, f is limited by a bound in Lemma 1. As a result, the complexity of the proposed approach is $O\left((\mu^{max})^3\alpha(f)^2 + \left\lfloor \frac{exec^{max}}{\min_{\forall k}(\delta_{I_f} + H_{I_f})} \right\rfloor\right)$, i.e., pseudopolynomial.

VI. Analysis for Superblocks and Tasks

In the previous section we showed how to derive the WCRT of a phase. This section describes how to compute the WCRT of a task, which is composed as a sequence of superblocks. Superblocks are composed of sequences of phases, or are represented by a single phase, depending on the resource access model, see Fig. 1.

The analysis is performed for each phase, while the worst-case starting time of a subsequent phase is the completion time of its preceding phase (in the subsequent execution model), or a dedicated starting time (in the time-triggered execution model). The starting time of the first phase in superblock $s_{i,j}$ is the worst-case completion time of its preceding superblock $s_{i-1,j}$, or a dedicated starting time in the subsequent and time-triggered execution model, respectively. For the first phase of the first superblock, the starting time is 0. If the deadline $d_{i,j}$ of a superblock $s_{i,j}$ is violated, then the task in unschedulable with this arbitration policy.

Consider $t_{i,j}$ to be the starting time of superblock $s_{i,j}$ (e.g., in the time-triggered execution mode), then without the need to consider any previous or subsequent superblock, the analysis can be performed accordingly. In case the resulting completion time exceeds deadline $d_{i,j}$, the task is unschedulable.

The pseudo-code for analyzing a task under the sequential model is given in Algorithm 2. Extension to the time-triggered model is omitted due to space constraints. Consider the set of superblocks $\mathcal{S}_j$ statically scheduled on processing element p_j and the set of interfering elements to be in IP. Furthermore, we assume that the length of an arbitration round $\Delta_\Theta + L_\Theta$ and the period W_j of the set of superblocks $\mathcal{S}_j$ under analysis to be integer multiples. Otherwise, Algorithm 2 would have to iterate over all possible offsets, i.e., the lcm of $\Delta_\Theta + L_\Theta$ and W_j.

Algorithm 2 Analyze Task

```
procedure AnalyzeTask($\mathcal{S}_j, IP$)
    $schedulable = true$;
    $t = 0$;
    for each $s_{i,j} \in \mathcal{S}_j$ do
        for each phase in $s_{i,j}$ do
            $exec^{max}$ max. amount of computation
            $\mu^{max}$ max. amount of resource accesses
            $t = t+$AnalyzePhase($t$, $exec^{max}$, $\mu^{max}$, $\Theta$, $IP$);
        end for
        if $t > d_{i,j}$ then
            $schedulable = false$;
        end if
    end for
    return $schedulable, t$
end procedure
```

Theorem 1: Algorithm 2 computes the worst-case response time (WCRT) for a set of superblocks $\mathcal{S}_j$ executing sequentially on a processing element p_j.

Proof: The proof is based on Lemma 4 and the assumption of a fully timing compositional architecture, i.e., no timing anomalies. Activating a phase at time t_1 and t_2 results in $WCRT_1 \leq WCRT_2$ for $t_1 \leq t_2$. Therefore a sequence of phases can be analyzed by setting the start time of a phase as its predecessors completion time, or 0 if it is the first phase. ■

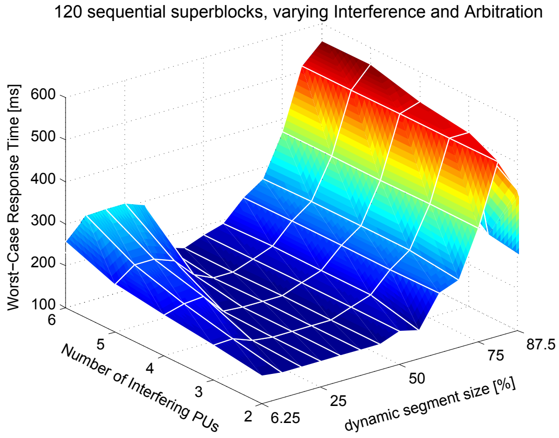
120 sequential superblocks, varying Interference and Arbitration

Fig. 4. Results for an automotive application.

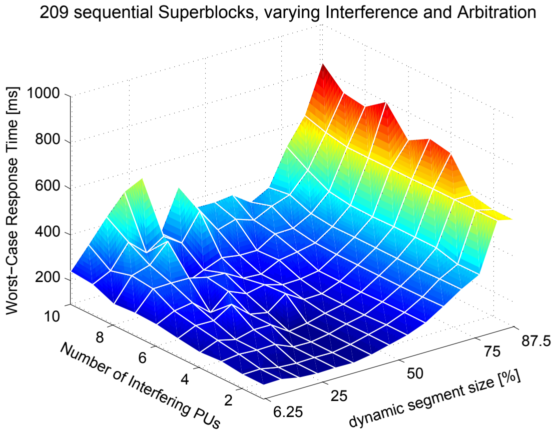
209 sequential Superblocks, varying Interference and Arbitration

Fig. 5. Results for a generated application.

VII. Simulations

We present experimental results based on a real-world application and on a generated application. The applications are composed of a set of subsequent superblocks, that are modeled according to the general model. The industrial application, from the automotive domain, is composed of 120 superblocks[3]. The interference on the shared resources is modeled as an arrival curve α_j for each interfering element, and assumed with a constant slope. The generated application is composed of 209 superblocks, using parameters extracted from the industrial application. It issues 230 access requests and performs computation amounting for about 9 ms. The time required to serve an access request, once access to the resource is granted, is 0.5ms, for both applications. The configuration of the arbiter is varied, such that 6.25% to 87.5% of an arbitration cycle belongs to the dynamic segment, and conclusively 93.75% to 12.5% belongs to the static segment, respectively. In the static segment, one slot is assigned to the application under analysis, while the remainder of the static segment is assigned to other elements, and thus unavailable. This slot amounts for 62.5% of the static segment length. Varying sizes of the dynamic

[3] Due to confidentiality agreements, data samples cannot be published

segment affect the size of the static segment, since a larger dynamic segment implies a smaller static segment.

In Fig. 4, an automotive application is analyzed. For an arbiter with only a small dynamic segment (6.25%), the WCRT increases from 166ms, for 2 interfering processing elements, to 260ms, for a system with 6 interfering elements (which equals an increase of $260/166 - 1 = 56.6\%$). With increasing share of the dynamic segment in the arbiter, the WCRT rises for systems with a high number of interfering elements. A dynamic segment, amounting for 25% of the arbitration cycle results in a WCRT of 161ms for two interfering elements, but in a WCRT of 309ms for a system with 6 interfering elements. This equals an increase of $309/161 - 1 = 91.9\%$.

Further expansion of the dynamic segment, for a large amount of interferers, at first results in a better performance, i.e., the WCRT decreases again. At some point, this effect reverses, and the performance suffers. For the application under consideration, a dynamic segment amounting for 25% to 50% of the arbitration cycle results in the lowest WRCT, i.e., the best performance. For larger dynamic segments, the performance suffers from the diminishing share of the static slot. The smaller the static slot, i.e., the guaranteed service, the larger the WCRT, even for systems with only few interfering elements. In Fig. 4, the increase of the WCRT for dynamic segments of 50% to 75% is apparent. For a system with only two interfering elements the WCRT increases from 161ms to 514ms, respectively (equals an increase of $514/161 - 1 = 219\%$.)

Further expansion of the dynamic segment resulted in another reduction of the WCRT. This is a result of (a) only little interference on the shared resource due to the small amount of interfering elements, and (b) the reduced influence of the blocking time in the static segment, between the end of the static slot (aka. frame) assigned to the application under analysis and the activation of the dynamic segment. With the static segment getting smaller and smaller, also this blocking time gets less significant.

Similarly to the automotive application, the generated application in Fig. 5 illustrates the same effects. An increasing amount of interference also results in an increased WCRT, but the effect is reduced compared to Fig. 4. The amount of interference that is suffered during the dynamic segment depends on the number of interfering elements. In our experiments, we assumed the overall amount of interference to be the same as for the automotive application.

Experimental results show that the problem is non-convex and an accurate WCRT is very hard to estimate. Many parameters, such as the size of the respective segments of the arbiter, the amount and pattern of interference and the number of interfering elements have an impact on the WCRT of an application. For that reason, an efficient and tight analysis is even more essential, since small changes in the design might lead to a significant increase of the WCRT.

VIII. Conclusion

In this paper, we propose an analysis approach to derive a safe (upper) bound of the worst-case response time (WCRT) for tasks in a resource sharing multiprocessor system. Tasks are composed of superblocks and phases, and are statically scheduled on the processing elements. Arbitration on the shared resource is based on an adaptive protocol, e.g., FlexRay protocol in automotive applications, where a static segment with fixed assigned time slots (TDMA) is succeeded by a dynamic segment with FCFS arbitration. Access to the shared resource is non-preemptive and synchronous and results in blocking of the task until access is granted by the arbiter. We propose a computationally efficient analysis approach that takes interference due to contention on the shared resource into consideration. Using a dynamic programming approach, we derive a tight WCRT, for a given tight interference representation. Experimental results show that the problem cannot be modeled as a convex optimization problem and that minor deviations result in significantly increased WCRT. Extensions to adaptive arbiters with greedy arbitration and Round Robin arbitration in the dynamic segment are possible, but omitted due to space constraints.

References

[1] AutoSAR. Release 4.0, http://www.autosar.org.

[2] D. B. Chokshi and P. Bhaduri. Performance analysis of FlexRay-based systems using real-time calculus, revisited. In *SAC*, pages 351–356, 2010.

[3] S. A. Edwards and E. A. Lee. The case for the precision timed (pret) machine. In *DAC*, pages 264–265, 2007.

[4] FlexRay. http://www.flexray.com/.

[5] R. Graham. Bounds on the performance of scheduling algorithms. In *Computer and Job Scheduling Theory*, pages 165–227. John Wiley and Sons, 1976.

[6] N. Guan, M. Stigge, W. Yi, and G. Yu. Cache-aware scheduling and analysis for multi-cores. In *EMSOFT*, October 2009.

[7] K. Lakshmanan, G. Bhatia, and R. Rajkumar. Integrated end-to-end timing analysis of networked autosar-compliant systems. In *DATE*, 2010.

[8] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury. Timing analysis of concurrent programs running on shared cache multi-cores. In *RTSS*, 2009.

[9] R. Pellizzoni and M. Caccamo. Impact of peripheral-processor interference on WCET analysis of real-time embedded systems. *IEEE ToC*, 59:400–415, 2010.

[10] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *DATE*, 2010.

[11] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39(1-3):205–235, 2008.

[12] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, pages 49–60, 2007.

[13] S. Schliecker, M. Negrean, and R. Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *DATE*, Dresden, Germany, mar 2010.

[14] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst. Reliable performance analysis of a multicore multithreaded system-on-chip. In *CODES/ISSS*, pages 161–166, 2008.

[15] A. Schranzhofer, J.-J. Chen, and L. Thiele. Timing analysis for tdma arbitration in resource sharing systems. In *RTAS*, 2010.

[16] Symtavision. SymTA/S Toolbox. http://www.symtavision.com/symtas.html.

[17] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis: a case study. *STTT*, 9(6):649–667, Nov 2006.

[18] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems. *IEEE TCAD*, 28(7):966–978, July 2009.