Timing Analysis for TDMA Arbitration in Resource Sharing Systems

Andreas Schranzhofer, Jian-Jia Chen, Lothar Thiele
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
Email: {schranzhofer,jchen,thiele}@tik.ee.ethz.ch

***Abstract*—Modern computing systems have adopted multi-core architectures and multiprocessor systems on chip (MP-SoCs) for accommodating the increasing demand on computation power. However, performance boosting is constrained by shared resources, such as buses, main memory, DMA, etc. This paper analyzes the worst-case completion (response) time for real-time tasks when time division multiple access (TDMA) policies are applied for resource arbitration. Real-time tasks execute periodically on a processing element and are constituted by sequential superblocks. A superblock is characterized by its accesses to a shared resource and its computation time. We explore three models of accessing shared resources: (1) dedicated access model, in which accesses happen only at the beginning and the end of a superblock, (2) general access model, in which accesses could happen anytime during the execution of a superblock, and (3) hybrid access model, which combines the dedicated and general access models. We present a framework to analyze the worst-case completion time of real-time tasks (superblocks) under these three access models, for a given TDMA arbiter. We compare the timing analysis of the three proposed models for a real-world application.**

Keywords: Worst-Case Timing Analysis, Time Division Multiple Access (TDMA), Real-Time Systems.

I. Introduction

Multiprocessor systems on chip (MPSoCs) and multicore platforms have been widely applied for modern computer systems to reduce production cost without sacrificing performance or significantly increasing power consumption. Multiple processing elements, working collaboratively on a common task, increase the computation power. Shared resources, such as buses, main memory, and DMA in multicore and MPSoC systems, and communication peripherals to connect nodes in distributed systems now represent the bottleneck for performance and timing predictability. Multiprocessor and MPSoC systems are typically designed to improve the average-case performance, while worst-case timing guarantees are usually not taken into consideration. However, guarantees on worst-case response/completion times are key requirements when implementing hard real-time applications, such as avionic and automotive applications.

Consider a platform with multiple processing elements and a single shared main memory. Executing a task on a processing element requires fetching of program instructions and acquisition of data from main memory. Moreover, communication among tasks on different processing elements also incurs memory access. As a result, contention on shared resources in general, and on main memory in particular, significantly delays the completion of tasks. Even though multiprocessor scheduling has been widely studied in the literature, e.g., [4], [5], [15], timing analysis for systems with shared resources has recently become an important topic. In particular, Edwards and Lee [2] propose modifications to either memory arbitration or cache behavior to improve timing predictability. However, the approaches in [2] require architectural modifications, which are not applicable for most commercial platforms.

For systems with shared resources, Pellizzoni et al. [9], [10], Negrean et al. [8] and Schliecker et al. [12], [13] have recently proposed methodologies to analyze the worst-case delay a task suffers due to accesses to a shared bus and shared memory. In [9], [10] a real-time task, that executes on a processing element, and peripherals share a bus, connecting them to a shared memory. The authors develop a framework to analyze the delay the task may suffer due to peripheral interference. Schliecker et al. [12], [13] and Negrean et al. [8] assume a set of tasks executing on a set of processing elements, all accessing a global shared resource. Event models are used to specify a tasks minimum and maximum number of accesses in a particular time window. The worst-case interference is derived by considering interferences from higher priority tasks. An iterative approach is used to propagate these interferences to lower priority tasks and to compute the worst-case delay. Priorities are assigned statically, and therefore interferences on a task propagate to all lower priority tasks.

Other works, closely related to interferences on shared resources, focus on interference caused due to cache accesses, e.g., Guan et al. [6] and [17], and Li et al. [7]. In [6], the authors propose a scheduling algorithm and a schedulability test based on linear programming for multiprocessor systems with shared L2 cache, while timing analysis is provided in [7]. In [17], the authors focus on the analysis of the worst-case execution time for a limited hardware model with only a single real-time task.

Despite work that analyzes the interference on shared resources and the delays suffered due to that, another research

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 216008 project Predator and the European Network of Excellence on Embedded System Design ARTISTDesign.

direction focuses on different arbitration policies for shared resources, such that interference is eliminated. One example for such an arbitration policy is time division multiple access (TDMA), where the shared resource is assigned to a task or processing element for a particular time window. In other words, access to the resource is partitioned over time and only a single actor can acquire it at a time. TDMA arbitration policies are often used in industrial applications, not only to increase timing predictability but also to alleviate schedulability analysis. Rosen et al. [11] and Andrei et al. [1] show approaches how efficient TDMA arbitration policies can be obtained. In these works, static analysis is used to compute the set of feasible traces a task can produce. The resulting traces are then applied to a given TDMA arbiter and the worst-case response time is chosen as the maximum among the traces' execution times.

Contrary to that, we assume that the positions of accesses to the shared resource are not known a priori and neither is their order. Producing all the feasible traces for such a configuration would result in an infinite number of possibilities. We assume a hardware platform where execution time and communication time can be decoupled, e.g., the fully timing compositional architecture proposed by Wilhelm et al. [16]. That is, we implicitly consider a hardware platform that has no timing anomalies. Based on this assumption, we provide an efficient and safe worst-case response time analysis, without the need to derive a large number of possible execution traces.

This paper considers systems with a shared resource, that requires a constant amount of time to complete a request. Access to the resource, e.g., a bus in a multicore system, is granted for at most one request at a time, resulting in blocking/waiting time for any other request. An ongoing access to a shared resource cannot be preempted.

We consider systems with real-time tasks under a given task partitioning, in which a task is allocated on a predefined processing element. Each processing element executes multiple statically scheduled tasks, and the static schedule is repeated periodically. Tasks considered in this paper are specified by sequential non-preemptable superblocks. Superblocks are specified with their maximum required computation time and their maximum number of accesses to the shared resource. Dedicated phases at the beginning and the end of each superblock are employed as *acquisition* and *replication phase* respectively. After acquiring required data from the shared resource, computations can be performed, i.e., the execution phase starts. This phase is then followed by the replications phase. For example, for accessing data from the main memory, the superblock reads the required data from the main memory in the acquisition phase, continues to the execution phase once all data is acquired and writes modified data back to the main memory in the replication phase.

We explore three models to specify access to the shared resource. Each model has a different degree of uncertainty and represents a trade-off between degree of freedom in the design process and the accuracy of the worst-case timing analysis:

Dedicated access model: Access to the shared resource is limited to the acquisition phase and the replication phase and is defined as upper bound for each superblock. The execution phase does not require any access to the shared resource. As shown in [3], this model of accessing shared resources can be mapped to an AutoSAR system for automotive applications.

General access model: Dedicated phases are omitted, and therefore accesses can happen at any point in time. Acquisition and replication happen on demand, during the active time of a superblock. This model considers the uncertainty due to different program paths or synchronous task communication.

Hybrid access model: Access to the shared resource can happen in any of the three phases and upper bounds on the number of accesses are specified. This model considers specified resource access patterns as in the dedicated access model and the uncertainty of the general access model.

The contributions of this paper are as follows:

- For the considered access models and a given TDMA arbiter, we propose an analytical worst-case analysis framework, considering blocking/non-buffered access to a shared resource.
- For the dedicated access model, our proposed analysis framework is of linear time complexity.
- For a regular TDMA arbiter, in which each processing element has only one time slot in a TDMA cycle, we show the efficiency of our analysis.
- For TDMA arbiters without constraints, we show that the timing analysis can be done by applying binary search.

The rest of the paper is organized as follows: In Section II, we introduce the proposed task model and the three different models of accessing a shared resource. We show the analytical framework to compute the worst-case completion time under a given TDMA arbiter in Section III for a phase and in Section IV for superblocks and tasks. Experimental results using a real-world application are shown in Section V. Section VI concludes the paper.

II. System Model

This section presents the system model we consider in this paper, including 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

A system is composed of multiple processing elements $p_j \in \mathcal{P}$. The processing elements in $\mathcal{P}$ execute independently, but share a common resource, e.g., an interconnection

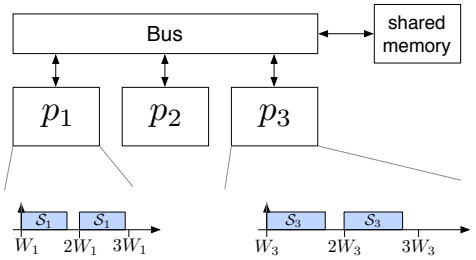


Figure 1. System Overview and Task Model

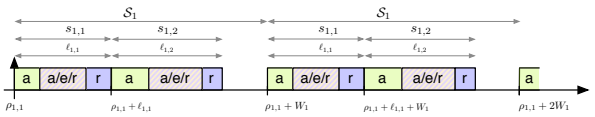
fabric (bus) to access a shared memory, see Fig. 1. We assume a given task partition, in which task set $\mathcal{T}_j$ is assigned to execute on $p_j \in \mathcal{P}$. A task is constituted by a sequence of superblocks. Superblocks execute sequentially, i.e., a succeeding superblock is not allowed to activate before its preceding superblock has finished.

We assume a static schedule among the superblocks by ordering the superblocks on a processing element in a predefined order. One might assume non-preemptive scheduling on task level by executing superblocks of a task sequentially, or might assume preemptive scheduling on task level by mixing the execution of superblocks of different tasks. However, a superblock is the basic unit for execution, in which preemption is not allowed. Otherwise, the profiled characteristics for the worst-case analysis in terms of execution time and resource accesses could be invalidated.

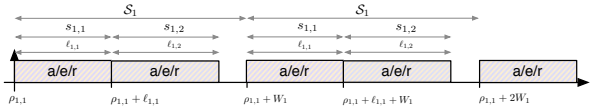
We consider a (given) repeating schedule of length W_j time units, denoted as *processing cycle*, on processing element p_j, in which the first superblock starts at time 0. That is, the schedule gives an order of the superblocks $\mathcal{S}_j$ constructed by the tasks assigned on p_j. Superblocks $\mathcal{S}_j$ are executed in time interval $(0, W_j]$, and are then repeated in $(W_j, 2W_j]$, $(2W_j, 3W_j]$, etc. Tasks have their specified earliest starting time, and thus superblocks cannot start before their corresponding tasks' earliest starting times. Following that, we define a superblock $s_{i,j}$ with an earliest starting time $\rho_{i,j}$ in a time window of length W_j and its *relative deadline* $\ell_{i,j}$. That is, to meet the timing constraint, an instance of superblock $s_{i,j}$ released at time $gW_j + \rho_{i,j}$ must be finished no later than $gW_j + \rho_{i,j} + \ell_{i,j}$, where the completion time minus the release time is the *response time* of the superblock. For simplicity, we assume that the deadline of a superblock released in a processing cycle on p_j is no more than the end of the processing cycle.

Superblocks are further structured in phases: *acquisition phase*, *execution phase*, and *replication phase*. This paper considers three models to specify these phases and accesses to the shared resource. Each model represents a trade-off between design freedom and accuracy of timing analysis.

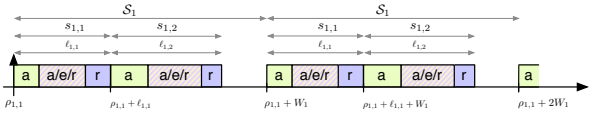
Dedicated access model: Accesses to the shared resource are limited to the acquisition phase at the beginning of the superblock and to the replication phase at the end of the superblock. After the activation of a superblock,



(a) Dedicated access phases: access to the shared resource only in (a)cquisition and (r)eplication phases



(b) General access phases: access to the shared resource anytime



(c) Hybrid access phases: access to the shared resource in (a)cquisition, (r)eplication, and (e)xecution phases

Figure 2. Overview of models to access the shared resource

requests to the shared resource are issued, in order to receive required data. In the succeeding execution phase, the actual computation takes place, while accesses to the shared resource are prohibited. After results are computed, the replication phase is used to update the corresponding data on the shared resource. Once the replication phase is finished, the superblock completes, see Fig. 2(a) for an example. Requests to the shared resource, as well as the time required for computations, are specified as upper bounds. The parameters for superblock $s_{i,j}$ are:

- $\mu_{i,j}^{max,a}$: max. number of requests in acquisition phase,
- $\mu_{i,j}^{max,r}$: max. number of requests in replication phase, and
- $exec_{i,j}^{max}$: max. execution time excluding resource accesses.

We assume these parameters were derived either by profiling and measurement, as shown by Pellizzoni et al. [9], or by applying static analysis for scratchpad memory to fetch new memory sections, etc.

General access model: Accesses to the shared resource are not limited to specific phases and can happen at any time and in any order. Conclusively, $\mu_{i,j}^{max,a}$ and $\mu_{i,j}^{max,r}$ are both 0, while parameter $\mu_{i,j}^{max,e}$ defines the maximum number of requests during the superblocks active time, see Fig. 2(b) for an example. The execution time is bounded by $exec_{i,j}^{max,e}$.

We assume $\mu_{i,j}^{max,e}$ is given by the designer either by profiling as in Pellizzoni et al. [9], applying static analysis for scratchpad memory, or by applying static analysis for cache misses, incurring accesses to the shared resource, i.e., the main memory [14], etc.

Hybrid access model: Accesses to the shared resource can happen in the acquisition, the execution, and the replication phases. This model allows to access the shared resource outside the dedicated acquisition and replication phases, e.g., reloading altered data during the execution phase. Requests

to the shared resource during the execution phase can happen anytime and are constrained by upper bounds. See Fig. 2(c) as an example. As a result, the parameters are $\mu_{i,j}^{max,a}$, $\mu_{i,j}^{max,r}$, $\mu_{i,j}^{max,e}$, and $exec_{i,j}^{max}$.

Without loss of generality, the hybrid access model covers the definitions of the dedicated model and the general access model, by modeling the first one with $\mu_{i,j}^{max,e} = 0$, while modeling the later one with $\mu_{i,j}^{max,r} = \mu_{i,j}^{max,a} = 0$.

Applying the models: The presented access models allow designers to tradeoff analyzability and predictability with restrictions on the model of programming. In Section III, we show how to analyze these models and show which restrictions have to be applied in order to provide a tight analysis. As an example, for applications focusing on control or signal processing, the *dedicated phases access model* is well suited. These applications are typically very predictable with respect to time and frequency that a resource has to be accessed (e.g., a particular sensor). Other applications, that require user input or event triggered data cannot be designed according to this model, and the *general access model* needs to be applied. The *hybrid access model* allows to increase the analyzability of a design, by enriching the *general access model* with dedicated access phases. In our analysis, we show that additional restrictions need to be applied on the shared resource arbitration in order to derive an efficient analysis methodology for the later two models.

Platform Assumptions: In this paper, we assume a hardware platform that allows to derive measures for execution time and the number of accesses to a shared resource without anomalies and decoupled from each other. Furthermore, requests to a shared resource, once access is granted, can be performed in (or bounded by) a constant amount of time.

B. Model of the Shared Resource

This paper considers systems with a stateless shared resource among the processing elements in $\mathcal{P}$. Any request to the shared resource has to wait until it is granted by the resource arbiter. After a request is granted, the shared resource starts to serve the request. At any time, the shared resource can serve at most one request. After the resource arbiter grants access of a request to the shared resource, the accessing time to the shared resource is (bounded by) a *constant* C. Therefore, if a superblock $s_{i,j}$ can access the shared resource at any time, the maximal time for executing the superblock $s_{i,j}$ is $exec_{i,j}^{max} + (\mu_{i,j}^{max,a} + \mu_{i,j}^{max,e} + \mu_{i,j}^{max,r}) \cdot C$. The shared resource is assumed non-preemptive, and hence, the resource arbiter only grants access to a pending request, when there is currently no other request served by the shared resource. Moreover, resource accesses in this paper are assumed *non-buffered*, such as shared memory access due to cache misses. This means that a task has to stall until its request to the shared resource is served successfully.

This paper considers systems with a TDMA arbiter for arbitrating access to the shared resource. A TDMA schedule

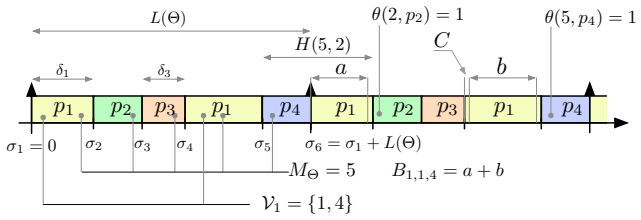


Figure 3. TDMA Scheme Overview

Θ is defined as sequence of time slots, such that $\sigma_m \in \Theta$ is the starting time of the m-th time slot (a.k.a. time slot m) and its duration is $\delta_m = \sigma_{m+1} - \sigma_m$. For notational brevity, we define M_Θ as the number of slots in schedule Θ and $L(\Theta)$ as its length. As a result, the TDMA schedule is repeated after every $L(\Theta)$ time units. For notational brevity, we set σ_1 as 0 and $\sigma_{M_\Theta+1}$ as $L(\Theta)$. Time slots of the schedule are assigned to processing elements, such that $\theta(m, p_j) = 1$ if time slot m is assigned to processing element p_j, otherwise $\theta(m, p_j) = 0$. For example, Fig. 3 illustrates an example for the TDMA schedule, while undefined variables will be explained later in Section III-A.

Only requests from processing element p_j with $\theta(m, p_j)$ equal to 1 are granted access to the shared resource in the m-th time slot of a TDMA cycle, while requests from other processing elements have to wait. Moreover, if at a time t the remaining time $\sigma_{m+1} - t$ of a time slot cannot serve a new request, i.e., $\sigma_{m+1} - t \leq C$, the TDMA arbiter will not grant any requests until the time slot expires. Therefore, without loss of generality, and in order to provide meaningful results, we assume that slots in the schedule are at least of length C and there is at least one slot for each processing element $p_j \in \mathcal{P}$. Slots with a length less than C cannot serve any accesses.

We assume that processing elements and the resource arbiter initialize synchronously, such that the first slot on the shared resource and the first superblock on each processing element start at time 0. A TDMA schedule for the shared resource is said to be *schedulable* if all the superblocks/tasks in all processing elements can finish before their respective deadlines, i.e., the response time of a superblock is no more than the relative deadline. An access to the shared resource blocks all other requesting tasks/superblocks, until it is completed, and thus results in significantly increased worst-case response times. The objective of this paper is to provide efficient analysis methods to compute the worst-case response times for all the tasks on a resource sharing system, assuming TDMA arbitration of the shared resource.

III. Worst-Case Completion Time of A Phase

Given a TDMA schedule Θ, this section presents how to analyze the worst-case completion time of an acquisition/replication phase starting at a specific time t. The delay suffered by the phase depends on its structure, i.e., how accesses to the shared resource are organized. We will

first present the required terminologies and notations in Section III-A with an example schedule presented in Fig. 3, and then show how to analyze the worst-case completion time of a phase in Sections III-B and III-C.

A. Terminologies and notations used for analysis

Let $\pi(t)$ be the time slot of the TDMA schedule at time t and $\sigma_{\pi(t)}$ be its starting time in the TDMA cycle, such that

$$\left\lfloor \frac{t}{L(\Theta)} \right\rfloor L(\Theta)+\sigma_{\pi(t)} \leq t < \left\lfloor \frac{t}{L(\Theta)} \right\rfloor L(\Theta)+\sigma_{\pi(t)+1}. \quad (1)$$

For brevity, the next time slot n of the current time slot m is defined by the closest next time slot n in the TDMA schedule. That is, if $n \leq m$, the next time slot n is in the next TDMA cycle of time slot m; otherwise, they are in the same cycle. Therefore, the time distance $H_{m,n}$ from time slot m to a next time slot n is defined as follows:

$$H_{m,n} = \begin{cases} \sigma_n - \sigma_m & \text{if } n > m \\ \sigma_n + L(\Theta) - \sigma_m & \text{otherwise,} \end{cases} \quad (2)$$

where Fig. 3 provides an example for `H(5,2)`.

The maximal number of completed requests for processing element p_j in TDMA time slot m is defined by $\lambda_{j,m}$, in which $\lambda^{max}_{j,m}$ is $\left\lfloor \frac{\delta_m}{C} \right\rfloor \theta(m, p_j)$, for acquisition/replication phases where no computation can happen. Similarly, for execution phases, since we are allowed to perform computation between requests, the number of completed requests for processing element p_j in a TDMA time slot m is $\lambda^{min}_{j,m} = \left\lfloor \frac{\delta_m - C}{C} \right\rfloor \theta(m, p_j)$. As a result, the maximal number of completed requests from time slot m to the next time slot n is denoted $\Lambda^{max}_{j,m,n}$ and is computed as:

$$\Lambda^{max}_{j,m,n} = \begin{cases} \sum_{k=m}^{n} \lambda^{max}_{j,k} & \text{if } n > m \\ \sum_{k=m}^{M_\Theta} \lambda^{max}_{j,k} + \sum_{v=1}^{n} \lambda^{max}_{j,k} & \text{otherwise.} \end{cases} \quad (3)$$

Computing $\Lambda^{min}_{j,m,n}$ follows suit. For notational brevity, we denote $\sum_{v=1}^{M_\Theta} \lambda^{max}_{j,v}$ by Λ^{max^*} and $\sum_{v=1}^{M_\Theta} \lambda^{min}_{j,v}$ by Λ^{min^*}, which is the maximal and minimal number of completed requests in a TDMA cycle $L(\Theta)$ respectively.

Assume a TDMA schedule Θ, with multiple time slots assigned to a processing element p_j. Then the ordered vector $\mathcal{V}_j$ contains the time slots assigned to p_j and each element $v_q \in \mathcal{V}_j$ is an integer between 1 and M_Θ, such that $v_1 < v_2 \cdots < v_{|\mathcal{V}_j|}$ and $\theta(v_q, p_j) = 1$. Let $\Delta_j(v_q)$ be the amount of time that processing element p_j has to wait for the next time slot, after time slot v_q has expired, then:

$$\Delta_j(v_q) = \begin{cases} \sigma_{v_{(q+1)}} - \sigma_{v_q+1} & \text{if } q < |\mathcal{V}_j| \\ \sigma_{v_1} + L(\Theta) - \sigma_{v_q+1} & \text{otherwise.} \end{cases} \quad (4)$$

The number of time slots assigned to processing element p_j from time slot m to its next time slot n (excluding time slot n) is defined as follows:

$$\psi_{j,m,n} = \begin{cases} \sum_{v=m}^{n-1} \theta(v, p_j) & \text{if } n > m \\ \sum_{v=m}^{M_\Theta} \theta(v, p_j) + \sum_{v=1}^{n-1} \theta(v, p_j) & \text{otherwise.} \end{cases} \quad (5)$$

The number of time slots in a TDMA cycle assigned to processing element p_j is denoted by Ψ^*_j, where $\Psi^*_j = \sum_{m=1}^{M_\Theta} \theta(m, p_j)$.

Superblocks cannot proceed performing computations as long as there are unserved requests to the shared resource. As a consequence, if there is an unserved request at the end of a time slot assigned to p_j, further executions stall until the next time slot becomes active and the request can be served. These stalls can be maximized by issuing accesses to the shared resource at the end of time slots. Hence, the time between consecutive time slots assigned to p_j is spent stalling, while active time slots are used to compute the unserved access issued at the end of the preceding time slot and to perform computations.

The time used performing computations on p_j from time slot m to its next time slot n, assuming that a single request to the shared resource is issued in each time slot assigned to p_j is denoted as $B_{j,m,n}$. Intuitively, time slot m is used to perform computations on p_j, until $C - \epsilon$ time units before the time slot expires (with $C > \epsilon > 0$), when an access to the shared resource is issued. The request cannot be completed anymore, and p_j stalls until the next time slot becomes active. Once the next time slot m' becomes active, the unserved access is completed, consuming a constant time C. Following that, computations are performed until $C - \epsilon$ units of time before time slot m' expires, such that an issued access request could not be served anymore. As a result, in time slot m' computations amount for $\delta_{m'} - (2C - \epsilon)$ time units. Therefore, we can derive $B_{j,m,n}$ as follows:[1]

$$B_{j,m,n} = \theta(n, p_j)(\delta_j - C) + \begin{cases} \sum_{v=m}^{n-1} \theta(v, p_j)\{\delta_v - 2C\}^+ & \text{if } n > m \\ \left\{ \begin{array}{l} \sum_{v=m}^{M_\Theta} \theta(v, p_j)\{\delta_v - 2C\}^+ \\ + \sum_{v=1}^{n-1} \theta(v, p_j)\{\delta_v - 2C\}^+ \end{array} \right. & \text{otherwise.} \end{cases} \quad (6)$$

See Fig. 3 for an example to compute $B_{1,1,4}$. Similarly, the maximum *time for performing computations*, denoted B^*, in a complete TDMA cycle of length $L(\Theta)$ considering the previously defined conditions, computes as:

$$B^* = \sum_{v=1}^{M_\Theta} \theta(v, p_j)\{\delta_v - 2C\}^+. \quad (7)$$

B. Worst-case completion time for an acquisition (replication) phase

This subsection presents how to analyze the worst-case completion time for an acquisition (replication) phase, starting at time t. Basically, we compute the time required to complete the access requests and how many time slots are required to do so.

Suppose that μ^{max} is the maximum number of requests to the shared resource for a phase. Algorithm 1 presents the

[1] If $z > 0$, function $\{z\}^+$ is z; otherwise $\{z\}^+$ is 0. We ignore ϵ in (6) since the value is meaningful only when it is very close to 0.

Algorithm 1 WCT-AR

Input: $\Theta, p_j, \mu^{max}, t, \Lambda$;
Output: Worst-case completion time;
1: let t' be the starting time of the next time slot after time t;
2: **if** $\left\lfloor \frac{t'-t}{C} \right\rfloor \theta(\pi(t), p_j) \geq \mu^{max}$ **then**
3: **return** $t + C\mu^{max}$;
4: **else**
5: $\mu' \leftarrow \mu^{max} - \left\lfloor \frac{t'-t}{C} \right\rfloor \theta(\pi(t), p_j)$;
6: $t' \leftarrow t' + \left\lfloor \frac{\mu'}{\Lambda^*} \right\rfloor L(\Theta)$; $\mu' \leftarrow \mu' - \left\lfloor \frac{\mu'}{\Lambda^*} \right\rfloor \Lambda^*$;
7: find the *closest next time slot* n^* such that $\Lambda_{j,\pi(t'),n^*} \geq \mu'$;
8: $t' \leftarrow t' + H_{\pi(t'),n^*} + C \cdot (\mu' - \Lambda_{j,\pi(t'),n^*-1})$;
9: **end if**

pseudo-code of the analysis for the worst-case completion time. Parameter Λ is a vector representing either Λ^{max} or Λ^{min}, which will be explained later. Let t be the current time and t' be the starting time of the next TDMA time slot, i.e., $t' = \left\lfloor \frac{t}{L(\Theta)} \right\rfloor L(\Theta) + \sigma_{\pi(t)+1}$. If the current time slot can serve the required requests μ^{max} to the shared resource, the worst-case completion time is simply $t + C\mu^{max}$ (see Step 2 to 3 in Algorithm 1). Otherwise, we consider to issue $\left\lfloor \frac{t'-t}{C} \right\rfloor \theta(\pi(t), p_j)$ requests in time slot $\pi(t)$, which implies that there are $\mu' = \mu^{max} - \left\lfloor \frac{t'-t}{C} \right\rfloor \theta(\pi(t), p_j)$ unserved requests (Step 5). In each TDMA cycle $L(\Theta)$, we can complete at most Λ^* requests. Therefore, we require at least $\left\lfloor \frac{\mu'}{\Lambda^*} \right\rfloor$ TDMA cycles to complete the remaining accesses. Then time t' is set to $t' + \left\lfloor \frac{\mu'}{\Lambda^*} \right\rfloor L(\Theta)$, while reducing μ' to $\mu' - \left\lfloor \frac{\mu'}{\Lambda^*} \right\rfloor \Lambda^*$ (Step 6). The remaining μ' accesses can then be completed in the current or next TDMA cycle. The algorithm goes through $[\sigma_{\pi(t')}, \ldots \sigma_{M_\Theta}, \sigma_1, \ldots, \sigma_{\pi(t')-1}]$ until the remaining requests μ' are finished. Therefore, if $\Lambda_{j,\pi(t'),M_\Theta} > \mu'$, we find the minimum $n^* \leq \pi(t)$ such that $\Lambda_{j,\pi(t'),n^*} \geq \mu'$; otherwise, we find the minimum $n^* > \mu'$ such that $\Lambda_{j,\pi(t'),n^*} \geq \mu'$ (Step 7). Following that, the worst-case completion time is calculated in (Step 8).

We denote Algorithm 1 as WCT-AR. The time complexity is $O(M_\Theta)$. It can be improved to $O(\log M_\Theta)$ by applying binary search in Step 7 when all $\Lambda_{j,m,n}$ for all m, n are calculated a priori.

Lemma 1: For an acquisition or a replication phase on processing element p_j with at most μ^{max} requests to the shared resource, Algorithm WCT-AR by setting Λ as Λ^{max} derives the worst-case completion time.

Proof: This is because of the definitions of the TDMA schedule Θ and the processing cycle W_j in p_j. As there is no interference, the worst-case completion time is the time when all the requests are served. ■

C. Worst-case completion time for an execution phase

This subsection presents how to analyze the worst-case completion time for an execution phase on processing element p_j when the phase starts at time t, where μ^{max} is the maximum number of requests to the shared resource and $exec^{max}$ is the maximal execution time for the execution phase. If μ^{max} is 0, the worst-case completion time is $t + exec^{max}$.

A request issued $C - \epsilon$ time units before the expiration of the current time slot cannot be served immediately, for any $\epsilon > 0$. For simplicity, we assume a request cannot be served in the current time slot if $\epsilon \geq 0$, sacrificing only little tightness.

The following lemma considers the case, that the current time slot is not assigned to processing element p_j.

Lemma 2: If $\theta(\pi(t), p_j) = 0$, either (1) only executing computations before the earliest next time slot for p_j or (2) issuing one request immediately results in the worst-case completion time.

Proof: Consider the other case, that one executes computations and then issues a request before the next time slot for p_j. It is clear that the computation can be postponed such that the completion time is larger. Therefore, one of the two cases in the statement results in the worst-case completion time. ■

Algorithm 2, denoted WCT-E, presents the pseudo-code of the analysis for the worst-case completion time. If the current time slot is not assigned to processing element p_j, i.e., $\theta(\pi(t), p_j) = 0$, we can either issue a request, and thereby cause the superblock to stall (Step 6), or perform computations till the next time slot assigned to processing element p_j (Step 8) becomes active. In the analysis, both cases are examined and the maximum completion time is reported as the worst-case completion time.

Therefore, for the rest, we only have to consider the case that $\theta(\pi(t), p_j)$ is 1. Again, let t' be initialized as the starting time of the next TDMA time slot, i.e., $t' = \left\lfloor \frac{t}{L(\Theta)} \right\rfloor L(\Theta) + \sigma_{\pi(t)+1}$. If executing all the required computations can be done before $t' - C - t$, the analysis executes computations first and then starts to access the shared resource at time $t + exec^{max}$. For this case, the problem is reduced to an acquisition or a replication phase, and hence, we can use Algorithm WCT-AR (Step 16 in Algorithm 2. Note that Λ^{min} has to be used to represent the number of access requests that can be served.). Otherwise, we perform computations for $\{(t' - C)) - t\}^+$ units of time and then issue one access request. As a result, the remaining computation time is $e' = exec^{max} - \{0, (t' - C) - t\}^+$ and the remaining worst-case requests are $\mu' = \mu^{max} - 1$. If we serve *exactly one request* in one time slot assigned to processing element p_j by injecting *unlimited* requests, we can find the time slot in which the remaining computation time becomes 0. That is, we find integers h and n^* such that

$$hB^* + B_{j,\pi(t'),n^*} \geq e' > hB^* + B_{j,\pi(t'),n^*-1}. \quad (8)$$

The estimated completion time t_c follows as $t_c \leftarrow t' + hL(\Theta) + H_{\pi(t'),n^*-1} + C + (e' - hB^* - B_{j,\pi(t'),n^*-1})$. Therefore, if the number of remaining requests is more than

Algorithm 2 WCT-E

Input: $\Theta, p_j, exec^{max}, \mu^{max}, t$;
Output: Worst-case completion time;
1: **if** $\mu^{max} = 0$ **then**
2: **return** $t + exec^{max}$;
3: **else**
4: **if** $\theta(\pi(t), p_j) = 0$ **then**
5: let t^* be the starting time of the next time slot assigned for p_j in Θ;
6: $R_1 \leftarrow$ WCT-E($\Theta, p_j, exec^{max}, \mu^{max} - 1, t^* + C$);
7: **if** $exec^{max} - t^* + t > 0$ **then**
8: $R_2 \leftarrow$ WCT-E($\Theta, p_j, exec^{max} - t^* + t, \mu^{max}, t^*$);
9: **return** $\max\{R_1, R_2\}$;
10: **else**
11: **return** R_1;
12: **end if**
13: **end if**
14: let t' be the starting time of the next time slot after time t;
15: **if** $(t' - C) - t \geq exec^{max}$ **then**
16: **return** WCT-AR($\Theta, p_j, \mu^{max}, t + exec^{max} + C, \Lambda^{min}$);
17: **end if**
18: $e' \leftarrow exec^{max} - \{0, (t' - C) - t\}^+$; $\mu' \leftarrow \mu^{max} - 1$;
19: find h and n^* such that $hB^* + B_{j,\pi(t'),n^*} \geq e' > hB^* + B_{j,\pi(t'),n^*-1}$;
20: $t_c \leftarrow t' + hL(\Theta) + H_{\pi(t'),n^*} + C + (e' - hB^* - B_{j,\pi(t'),n^*-1})$;
21: **if** $h\Psi_j^* + n^* < \mu'$ **then**
22: **if** t_c and $t_c + C$ are not in the same time slot **then**
23: let t_c be the beginning of the next time slot assigned for p_j after t_c ;
24: **end if**
25: **return** WCT-AR($\Theta, p_j, \mu' - h\Psi_j^*, t_c + C, \Lambda^{min}$);
26: **else**
27: **return** WCT-E-SUB($\Theta, p_j, exec^{max}, \mu^{max}, t, t_c$);
28: **end if**
29: **end if**

there are time slots between t' and the estimated completion time, i.e., $h\Psi_j^* + n^* < \mu'$, we move to time t_c. Then, we waste the next C time units in the current (or the following) time slot of t_c without computation or resource accesses, and apply Algorithm WCT-AR for the remaining accesses $\mu' - h\Psi_j^* + n^*$ (Step 25).

Steps 16 and 25 represent the cases where all computations are executed and only access requests to the shared resource remain to be finished. The worst-case completion time of the remaining accesses can be computed by Algorithm WCT-AR, considering that in an execution phase the issuing time of an access request can be postponed by executing computations for ϵ units of time. Therefore, the number of access requests that can be served in any sequence of time slots is represented by Λ^{min}. Furthermore, C units of time are allocated to serve one open access request that might be pending at the beginning of the next active time slot.

If $h\Psi_j^* + n^* \geq \mu'$, the estimated completion time t_c gives an upper bound of the worst-case completion time. To derive a tighter worst-case completion time, we continue with Algorithm WCT-E-SUB (Algorithm 3) by specifying $R_{ub} = t_c$ as the upper bound of the worst-case completion time. The basic idea in Algorithm WCT-E-SUB is to first find a lower bound of the worst-case completion time by

Algorithm 3 WCT-E-SUB

Input: $\Theta, p_j, exec^{max}, \mu^{max}, t, R_{ub}$;
Output: Worst-case completion time;
1: let t' be the starting time of the next time slot after time t;
2: find h and n^* such that $h\Psi_j^* + \psi_{j,\pi(t),n^*} = \mu^{max} > h\Psi_j^* + \psi_{j,\pi(t),n^*-1}$ for some integer h;
3: $e' \leftarrow exec^{max} - B^* \left\lfloor \frac{\mu'}{\Psi_j^*} \right\rfloor - B_{j,\pi(t'),n^*} - (t' - t - C)$;
4: $R_{lb} \leftarrow t' + hL(\Theta) + H_{\pi(t'),n^*} + e'$;
5: **return** the binary search result $t^\dagger$ by applying WCT-E-SUB-TEST() by setting $t^\dagger$ in the range $[R_{lb}, R_{ub}]$;

issuing a single request to the shared resource in each time slot from the current time slot. After all requests are served, the remaining computations are finished, which gives a lower bound R_{lb} (Step 4 in Algorithm 3). Now, we have an upper bound and a lower bound of the worst-case completion time, and apply binary search to find $t^\dagger$ between these bounds and test whether $t^\dagger$ is feasible or not.

The test, whether $t^\dagger$ is feasible or not, is shown in Algorithm 4. As $t^\dagger \geq R_{lb}$, we know that there are at least $\mu^{\max}$ time slots available for processing element p_j from time t to time $t^\dagger$.

The schedulability test distributes the μ^{max} requests among the time slots for processing element p_j in time interval $(t, t^\dagger]$, such that the time spent waiting for grants to the shared resource is maximized. If we issue a request in a time slot v_q in $\mathcal{V}_j$, we will experience at most $\Delta_j(v_q) + 2C$ waiting time to finish the request. Suppose that η_n is the number of time slots $v_q \in \mathcal{V}_j$ in time interval $[t, t^\dagger]$ (excluding the time slot at time $t^\dagger$). That is,

$$\eta_n = \begin{cases} \left\lceil \frac{\hat{t} - t}{L(\Theta)} \right\rceil & \text{if } \pi(t) \leq v_q < \pi(\hat{t}) \text{ or } \pi(\hat{t}) < \pi(t) \leq v_q \\ & \text{or } v_q < \pi(t) < \pi(\hat{t}) \\ \left\lceil \frac{\hat{t} - t}{L(\Theta)} \right\rceil - 1 & \text{otherwise,} \end{cases} \tag{9}$$

where $\hat{t} = \sigma_{\pi(t^\dagger)}$ is the starting time of the time slot at time $t^\dagger$. As we want to find the largest waiting time among those $\sum_{v_q \in \mathcal{V}_j} \eta_n$ time slots, we find those μ^{max} time slots with the largest waiting time (Step 4 to Step 12 in Algorithm 4, where $wait$ is a variable to indicate the waiting time without considering the time to complete any request). Then, we test whether $wait + 2C\mu^{\max} + exec^{max}$ is less than $t^\dagger - t$ (Step 13 to Step 17 in Algorithm 4) to test the feasibility.

Correctness and Complexity of Algorithm WCT-E: The time complexity is $O(M_\Theta)$ if Algorithm WCT-E-SUB is not required. As $R_{ub} - R_{lb}$ is bounded by $exec^{max}$, the time complexity of Algorithm WCT-E-SUB is $O(M_\Theta \log exec^{max})$, which is also polynomial. Therefore, the overall time complexity is $O(M_\Theta \log exec^{max})$.

Due to the space limitation, we will only sketch the proofs for correctness. Given an execution phase on processing element p_j starting at time t with $exec^{max}$ computation time and μ^{max} requests to the shared resource, consider a schedule $\mathbf{S}$:

Algorithm 4 WCT-E-SUB-TEST

Input: $\Theta, p_j, exec^{max}, \mu^{max}, t, t^\dagger$;
Output: Feasibility to finish by $t^\dagger$ for the worst case;

```
for each v_q ∈ V_j do
    derive η_n by (9);
end for
μ' ← μ^max; wait ← 0;
while μ' > 0 do
    find the index q* such that Δ_j(v_q*) is the maximal among those v_n ∈ V_j with η_n > 0;
    if η_q* > μ' then
        wait ← wait + μ'Δ_j(v_q*); η_q* ← η_q* − μ'; μ' ← 0;
    else
        wait ← wait + η_q*Δ_j(v_q*); μ' ← μ' − η_q*; η_q* ← 0;
    end if
end while
if wait + 2Cμ^max + exec^max ≤ t^† − t then
    return "feasible";
else
    return "infeasible";
end if
```

- It issues an access request $C - \epsilon$ time units before the end of each upcoming time slot assigned to processing element p_j. This way, access requests cannot be served in their current time slot anymore and the system stalls until its next time slot becomes active, at which time C units of time are consumed to execute the access request. (Note: for time slots shorter than $2C$, a request is issued immediately after the pending request has completed, resulting in the system to block.)
- It executes computations at all times when the system is neither stalling nor executing access requests.

Suppose that, in schedule $\mathbf{S}$, t_1 is the completion time for preforming computations and t_2 is the completion time of the last request. Let μ_{t_1} be the requests completed before t_1 in schedule $\mathbf{S}$.

Lemma 3: If t_1 and $t_1 + C$ are in the same time slot, let $\bar{t}$ be $t_1 + C$; otherwise, let $\bar{t}$ be C plus the starting time of the next time slot for the phase after t_1. If $t_2 > t_1$, the completion time by issuing $\mu^{\max} - \mu_{t_1}$ requests after time $\bar{t}$ is an upper bound of the completion time for the execution phase starting at time t, assuming Λ^{min} as the number of access requests that can be executed in any sequence of time slots.

Proof: We can prove this lemma by swapping 1 request with computations amounting for C units of time. Let ϕ^* be the phase with computations and requests according to the statement in this lemma. For any different phase ϕ, starting from the first time slot with a different number of requests, it is not difficult to see that one can always swap the computation and one request without shortening the completion time. After all, we can transform any ϕ to ϕ^* without shortening the completion time, which proves this lemma. ■

Lemma 4: If $t_1 \geq t_2$ and $|\mathcal{V}_j| = 1$, the estimated completion time t_c in Algorithm 2 (R_{lb} in Algorithm 4) is an upper bound of the worst-case completion time.

Proof: The proof is very similar to Lemma 3, and we omit this due to space limitation. ■

Lemma 4 states that it is not necessary to run a binary feasibility test in Algorithm WCT-E-SUB for a regular TDMA arbiter, in which each processing element has only one access time slot in a TDMA cycle, i.e., $|\mathcal{V}_j| = 1$. The time complexity for such a case becomes $O(1)$. However, to the best of our knowledge, for general cases, binary search to test feasibility performs best.

Lemma 5: If $t_1 \geq t_2$ and setting the completion time $t^\dagger$ as $t_{\mathbf{S}'}$ returns "feasible" by Algorithm WCT-E-SUB-TEST, the completion time of any schedule $\mathbf{S}'$ for the execution phase is no more than $t^\dagger$.

Proof: When Algorithm WCT-E-SUB-TEST returns "feasible", it is not difficult to see and prove that Algorithm 4 is based on a concrete trace such that the total *blocking* time in which the phase waits for the grants to the shared resource is the *largest*. If a schedule $\mathbf{S}'$ with completion time larger than $t^\dagger$ exists, it must create longer blocking times to wait for accessing the shared resource, which contradicts the largest blocking time of the trace considered in Algorithm 4. ■

Lemma 6: If $t_1 \geq t_2$ and $t^\dagger$ is the worst-case completion time, applying Algorithm WCT-E-SUB-TEST by setting the completion time $t^\dagger - \epsilon$ as $t_{\mathbf{S}'}$ returns "infeasible", for any $\epsilon > 0$.

Proof: The proof is similar to the proof in Lemma 5. ■

As a result, we conclude the analysis for the execution phase by the following theorem.

Theorem 1: The worst-case completion time derived from Algorithm WCT-E provides an upper bound of the completion time for any feasible execution bounded by computation time $exec^{max}$ and requests μ^{max} on processing element p_j.

Proof: This is based on the lemmas in this subsection. ■

IV. Timing Analysis for Superblocks and Tasks

With the analysis of the completion time of a phase, we can analyze the schedulability of the superblocks/tasks on processing element p_j. Testing the schedulability of tasks on processing element p_j requires to analyze for a time period equal to the least common multiplier (LCM) of p_j's period W_j and the schedules' length $L(\Theta)$. This way, all possible offsets between slots and superblocks are considered.

Since there is no interference of other processing elements when we analyze processing element p_j, it is clear that there is no anomaly. That is, if a superblock completes earlier than its worst-case completion time, this will not increase the worst-case execution time of the remaining superblocks. As a result, we have to consider the superblocks' worst-case execution time sequentially, see Algorithm 5. Its time complexity is $O(|\mathcal{S}_j| \frac{LCM(W_j, L(\Theta))}{W_j})$.

To conclude the analysis, the following theorem gives the correctness of Algorithm 5.

Algorithm 5 Schedulability-Test

Input: $\Theta, p_j, \mathcal{S}_j, W_j$;
Output: Schedulability of TDMA schedules

for $g \leftarrow 0$; $g < \frac{LCM(W_j, L(\Theta))}{W_j}$; $g \leftarrow g + 1$ **do**
  $t \leftarrow g \cdot W_j$;
  for each $s_{i,j} \in \mathcal{S}_j$ sequentially **do**
    if $t < g \cdot W_j + \rho_{i,j}$ **then**
      $t \leftarrow g \cdot W_j + \rho_{i,j}$;
    end if
    $t \leftarrow$ WCT-AR($\Theta, p_j, \mu_{i,j}^{max,a}, t, \Lambda^{max}$);
    $t \leftarrow$ WCT-E($\Theta, p_j, exec_{i,j}^{max,e}, \mu_{i,j}^{max,e}, t$);
    $t \leftarrow$ WCT-AR($\Theta, p_j, \mu_{i,j}^{max,r}, t, \Lambda^{max}$);
    if $t - (g \cdot W_j + \rho_{i,j}) > \ell_{i,j}$ **then**
      return "might be unschedulable" for p_j;
    end if
  end for
end for
return "schedulable" for p_j;

Theorem 2: A TDMA arbiter based on TDMA schedule Θ is schedulable for scheduling tasks/superblocks assigned on processing element p_j if Algorithm 5 returns "schedulable".

Proof: This comes directly from Lemma 1 and Theorem 1 along with the fact that the early completion of a superblock will not increase the worst-case execution time of the remaining superblocks. ■

V. Experimental Results

In this section, we present the results of our proposed worst-case analysis framework, applied to sequences of superblocks of different sizes and under varying arbiters. We also show a concrete execution trace to visualize the process of obtaining the worst-case response time. Our proposed worst-case analysis is applied to two sequences of superblocks. The first sequence represents a small task, with only 8 subsequent superblocks. The second sequence is constituted by 82 superblocks. The superblocks' parameters are generated using random number generators, following specifications provided by an industrial partner.[2]

Firstly, superblocks are generated following the *dedicated access model*, i.e., there are three phases and only the first and last one issue accesses to the shared resource. Secondly, we derive data for the *general access model* and the *hybrid access model* from those superblocks generated before.

Based on the dedicated access model, the general access model is derived by accumulating the acquisition and replication phases' accesses to a single phase in the general model. The hybrid model is derived by moving a certain share of each dedicated phases' accesses to the execution phase, while at the same time, the number of accesses for a superblock remains the same for each of the three access models.

Our proposed analysis framework assumes a given TDMA arbiter and therefore, in order to perform experimental evaluations, an arbiter was constructed. We assume multiple processing elements, as a random number between 2 and 9, and compute the share of time slots each processing element is assigned to, based on the number of superblocks that are mapped onto them. The mapping of superblocks onto the processing elements is done greedily. Based on these numbers, two arbiters can be computed: (1) a *regular TDMA arbiter*, where all the slots assigned to a particular processing element are equally spaced or (2) an *irregular TDMA arbiter* with randomly distributed slots, where multiple slots of varying lengths are assigned to a processing element.

In Fig. 4 we show the results for the regular TDMA arbiter. Two superblock sequences are analyzed, in 4 different access models. The most left pair of bars in Fig. 4 represents the naive worst-case execution time, considering only a constant amount of time consumed by each access to the shared resource, i.e., when the shared resource is always available. The next three pairs of bars show the results for different memory access models. In Fig. 5 the same superblock sequences are analyzed again, but this time with an irregular TDMA arbiter, as described before.

The dedicated access model outperforms the other access models with respect to worst-case response time. This is due to the limited variability inherent to the dedicated access model. This effect is very apparent in Fig. 5, but also observable in Fig. 4. For the regular arbiter used in the first set of experiments, the delays are dominated by waiting for the next period of the arbiter, while in the second set of experiments, the distance between the single time slots is the determinant measure.

In order to derive the worst-case response time for superblocks, different traces of requests to the shared resource have to be examined. In Fig. 6, we analyze a sample superblock with earliest release time $\rho = 0.5$ms. We show one trace that leads to the worst case response time (WCRT) (feasible Trace 2) and one that does not.

Trace 1 (feasible Trace 1) starts with performing computations at time 0.5 ms while Trace 2 (feasible Trace 2) issues an access to the shared resource and therefore stalls until the next time slot becomes available, at time 2.0 ms. Eventually, Trace 1 finishes its computations and the remaining accesses to the shared resource have to be issued. Since the slot is currently active, these accesses are completed and the superblock finishes at 5.7ms. Trace 2, on the other hand, computes until the end of the second slot before issuing another access. After a long stall block, the system continues to do computations for 0.2ms before finishing at 8.25ms, which results in a worst-case response time (WCRT) of 7.75ms. This shows, that very small deviations can result in large variances on the resulting WCRT.

VI. Conclusion and Future Work

In this paper, we introduce a novel methodology to analyze the worst-case response time of real-time

[2]Due to confidentiality agreements, actual data and company names cannot be published.

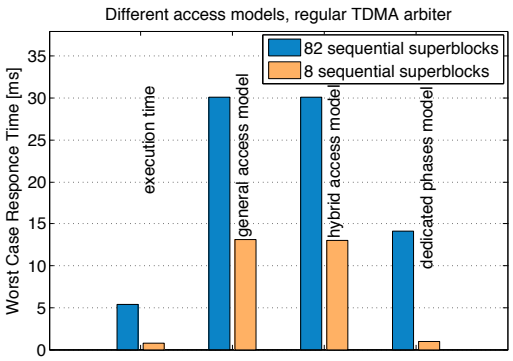


Figure 4. Experimental results for a regular TDMA arbiter.

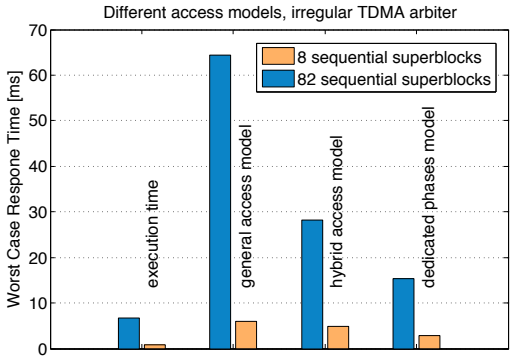


Figure 5. Experimental results for an irregular TDMA arbiter.

tasks/superblocks on systems with shared resources and TDMA arbitration polices. Accesses to the shared resource result in blocking of a real-time task/superblock until the request is completed, and therefore contention on the resource results in significantly increased delays. We present an analysis framework that allows to efficiently compute the worst-case response time and schedulability of tasks/superblocks specified according to any of the three proposed access models and for any TDMA arbitration on the shared resource.

We show that separating computations and accesses to the shared resource (dedicated access model) is of crucial importance in the design of resource sharing systems. Using this model allows to derive tight and efficient algorithms for schedulability analysis and significantly reduces the worst-case response time of real-time tasks.

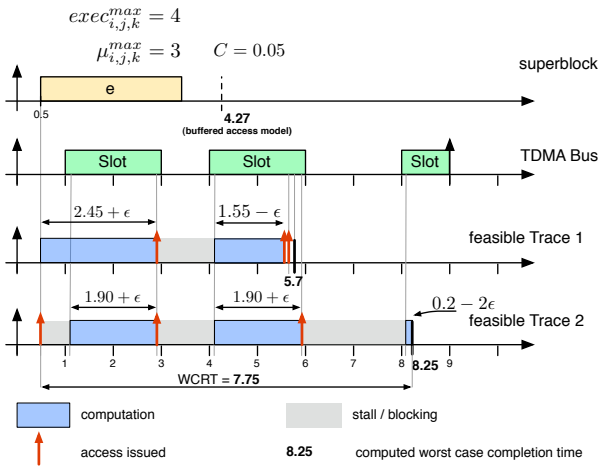


Figure 6. Worst-case completion time for a sample superblock and a TDMA arbiter (TDMA Bus).

References

[1] A. Andrei, P. Eles, Z. Peng, and J. Rosen. Predictable implementation of real-time applications on multiprocessor systems-on-chip. In *VLSID*, pages 103–110, 2008.

[2] S. Edwards and E. Lee. The case for the precision timed (PRET) machine. Technical Report UCB/EECS-2006-149, EECS Department, University of California, Berkeley, November 17 2006.

[3] A. Ferrari, M. D. Natale, G. Gentile, G. Reggiani, and P. Gai. Time and memory tradeoffs in the implementation of AUTOSAR components. In *DATE*, pages 864 – 869, Apr 2009.

[4] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[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*, pages 245–254, October 2009.

[7] 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.

[8] M. Negrean, S. Schliecker, and R. Ernst. Response-time analysis of arbitrarily activated tasks in multiprocessor systems with shared resources. In *DATE*, pages 524–529, April 2009.

[9] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of CPU and I/O transactions in COTS-based embedded systems. In *RTSS*, pages 221–231, Dec 2008.

[10] R. Pellizzoni and M. Caccamo. Impact of peripheral-processor interference on WCET analysis of real-time embedded systems. *IEEE TOC*, pages 400–415, March 2010.

[11] 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.

[12] S. Schliecker, M. Ivers, and R. Ernst. Integrated analysis of communicating tasks in MPSoCs. In *CODES+ISSS*, pages 288–293, 2006.

[13] 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.

[14] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separate cache and path analyses. *Real-Time Systems*, 18(2/3), May 2000.

[15] M. A. Trick. Scheduling multiple variable-speed machines. *Operations Research*, 42:234–248, 1994.

[16] 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. *TCAD*, 28(7):966–978, July 2009.

[17] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *RTAS*, pages 80–89, 2008.