# Timing Predictability on Multi-Processor Systems with Shared Resources

Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele

Computer Engineering and Networks Laboratory (TIK),
Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
{schranzhofer,jchen,thiele}@tik.ee.ethz.ch

**Abstract.** Multi-processor systems are becoming increasingly important in consumer electronics as well as in industrial applications, such as automotive software. Tasks need to share data across processing unit boundaries, e.g., local variables, triggering the need for a communication fabric. Therefore, multi-processor systems are constituted not only by a mere set of processing units, but also by communication and memory peripherals. These peripherals are *shared resources*, i.e., multiple independently executing tasks on multiple processing units compete for accessing them. Real-time tasks execute periodically on a processing element and are constituted by sequential superblocks. In this paper, we consider several models to schedule the superblocks and organize accesses to the shared resources within the superblocks. First, superblocks can be executed sequentially, i.e., a superblock is activated as soon as its preceding superblock has finished, or they can be executed according to a static schedule (preassigned time slots). Second, we consider three models to access 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 show the relation between these models with respect to schedulability and provide experimental results that show that the dedicated phases model with sequential superblocks performs best.

## 1 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. 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 significantly delays the completion of tasks.

For systems with shared resources, Pellizzoni et al. [2], Negrean et al. [1] and Schliecker et al. [4] have recently proposed methodologies to analyze the worst-case delay a task suffers due to accesses to a shared bus and shared memory. Another research direction focuses on different arbitration policies for shared resources to eliminate interference. For example, Rosen et al. [3] use TDMA for bus accesses.

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. Furthermore, 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. [5].

Consider a task being allocated on a predefined processing element. Tasks are constituted by superblocks, which are defined by their maximum required computation time and their maximum number of accesses to the shared resource. We consider different models to schedule superblocks on to specify accesses to shared resources therein. Superblocks execute either (1) sequentially or (2) time-triggered. First, sequentially executing superblocks start executing as soon as their preceding superblock has finished. The first superblock of a task starts execution as soon as the task is activated. Second, time triggered superblocks start execution at dedicated time instants statically.

Accesses to shared resources are specified according to three models: (1) dedicated phases model. (2) general model and (3) hybrid model. 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. The general model allows access to shared resources at any time during a superblock active time, i.e., the acquisition and replication phases merge with the execution phase. The hybrid model has an acquisition and an replication phase, but accesses to the shared resource can also happen during the execution phase. The goal of this study is to compare the worst-case response time for the above different models.

## 2 System Model

### 2.1 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 fabric (bus) to access a shared memory. 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:

**sequential** a succeeding superblock is not allowed to activate before its preceding superblock has finished or

**time triggered** a superblock starts execution at a predefined time and therefore does not depend on its succeeding superblocks' computation time.

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. 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. 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* $\ell_{i,j}$.

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, requests to the shared resource are issued, in order to receive required data. After results are computed, the replication phase updates the results. 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.

**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. However, requests are defined as an upper bound, as is the required time to compute results, see Fig. 1a. In addition to the definition of $exec_{i,j}^{max}$ in a superblock $s_{i,j}$, we also have
  - $\mu_{i,j}^{max,e}$: max. number of requests during the superblocks active time

**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. See Fig. 1e 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 $exec_{i,j}^{max}$.

### 2.2 Model of the Shared Resource

This paper considers systems with a TDMA arbiter for arbitrating the access to the share resource. A TDMA schedule $\Theta$ 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 $M_\Theta$ as the number of slots in schedule $\Theta$ 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 $\sigma_1$ as 0 and $\sigma_{M_\Theta+1}$ as $L(\Theta)$.

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. 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.

## 3 Schedulability Analysis

### 3.1 Access Models

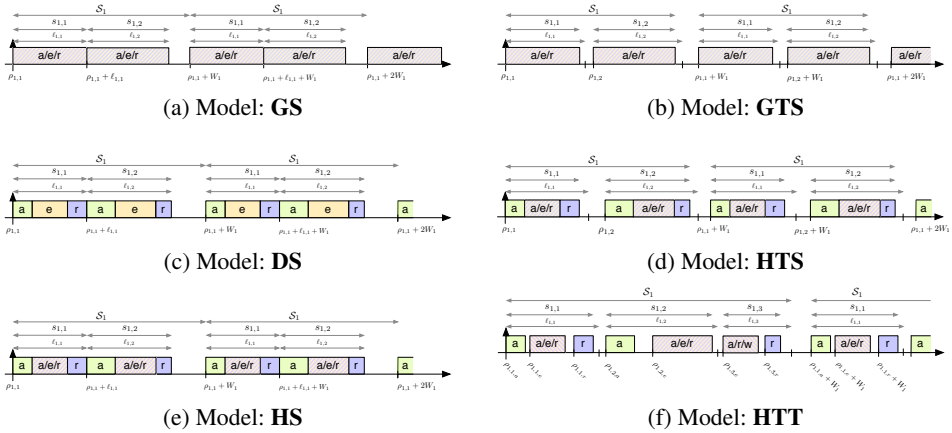
**sequential general model - GS** Superblocks execute sequentially, and accesses to the shared resource can happen anytime and in any order, see Fig. 1a.

**sequential dedicated model - DS** Superblocks execute sequentially, and accesses to the shared resource are in the acquisition and replication phases, see Fig. 1c.

**sequential hybrid model - HS** Superblocks execute sequentially, and accesses to the shared resource are issued in dedicated acquisition and replication phases. Additionally, in the execution phase, accesses to the shared resource can be issued at any time, see Fig. 1e.

**time-triggered superblocks general model - GTS** Superblocks start execution at dedicated points in time and accesses to the shared resource can happen at any time and in any order, see Fig. 1b.

**time-triggered superblocks hybrid model - HTS** Superblocks start execution at dedicated points in time and accesses to the shared resource are issued in dedicated acquisition and replication phases. Additionally, in the execution phase, accesses to




(a) Model: **GS** (b) Model: **GTS**

(c) Model: **DS** (d) Model: **HTS**

(e) Model: **HS** (f) Model: **HTT**

Fig. 1: The access models considered in this paper.

the shared resource can be issued at any time, see Fig. 1d. Phases of a superblock execute sequentially. As an example, the execution phase of superblock $s_{1,1}$ in Fig. 1d, starts after the acquisition phase has finished.

**time-triggered phases hybrid model - HTT** Superblocks are specified according to the hybrid model, i.e., there is an acquisition, an execution and a replication phase and each phase starts at a statically defined point in time. Accesses to the shared resource are issued in the acquisition, replication, and execution phases. In the latter, accesses are issued at any time and in any order, see Fig. 1f.

### 3.2 Schedulability Relationship

Assume a set of superblocks with defined maximum execution time and maximum number of access to the shared resource in each of its phases. Furthermore, assume this set of superblocks to be specified according to each of the six previously introduced models. We then test the schedulability of the set of superblocks, for each of the six specification models. Schedulability for some of the models can be derived from schedulability of other models, see Fig. 2:

- If the set of superblocks is schedulable for the *sequential general model*, then it is also schedulable for the *sequential hybrid model*. This is because the hybrid model is a specialization of the general model. In other words, each concrete execution trace that can be realized by the hybrid model can as well be realized by the general model.
- If the set of superblocks is schedulable for the *sequential hybrid model*, then it is also schedulable for the *sequential dedicated model*. The dedicated model is a special case of the hybrid model. Similarly to the previous relation, each trace produced by a set of superblock specified according to the dedicated model, can as well be produced by a set of superblock specified according to the hybrid model.
- If the set of superblocks is schedulable for the *time-triggered superblocks general model*, then it is also schedulable for the *time-triggered phases hybrid model*. First, the hybrid model is a special case of the general model. Second, time-triggered phases inside a superblock are a special case of the time-triggered superblocks models. The activation time of the phases in the time-triggered phases model has to

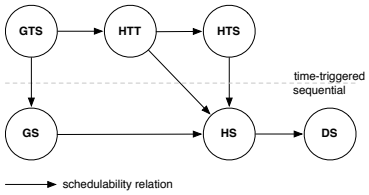


Fig. 2: Schedulability Relationship between different models.

be chosen, such that the preceding phase can finish in any case (worst-case). Therefore, each execution trace that can be produced by the time-triggered phases model, can as well be produced by the time-triggered superblocks model, by assuming the worst case completion time for each phase.

- If the set of superblocks is schedulable for the *time-triggered phases hybrid model*, then it is also schedulable for the *time-triggered superblocks hybrid model*. It is easy to see, that the time-triggered phases model is a special case of the time-triggered superblocks models. Furthermore, the model to access the shared resource is the same - namely the hybrid model. Consequently, this schedulability relation is valid.
- If the set of superblocks is schedulable for the *time-triggered superblocks general model* then it is also schedulable for the *sequential general model*. Similarly to previous relations, the time-triggered model is a special case of the sequential case.
- If the set of superblocks is schedulable for the *time-triggered phases hybrid model* or for the *time-triggered superblocks hybrid model*, then it is also schedulable for the *sequential hybrid model*. The time-triggered models are special cases of the sequential model and the model to access the shared resource is the same. Therefore, the schedulability relation is valid.

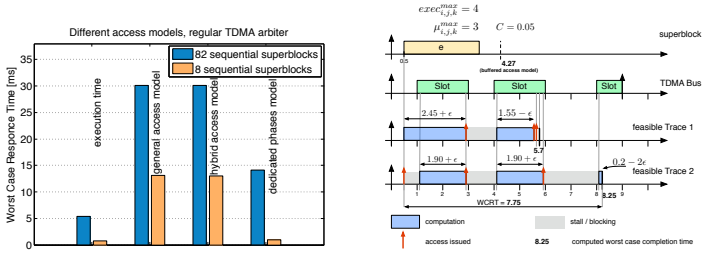
As a conclusion, if any of the six models results as schedulable, the *sequential dedicated model* is schedulable as well.

## 4 Experimental Results

In this section, we present the results for the sequential models, since the time-triggered models are a special case of them. We analyze two set of superblocks, specified according to the three different models to access the shared resource. 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.

Two superblock sequences are analyzed, in 4 different access models. The most left pair of bars in Fig. 3a 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. The dedicated access model outperforms the others 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. 3a.

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. 3b, 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 1) and one that does not.




(a) A task (b) A sample superblock.

Fig. 3: Experimental results for a regular TDMA arbiter

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 $5.7$ms. 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 $0.2$ms before finishing at $8.25$ms, which results in a worst-case response time (WCRT) of $7.75$ms. This shows, that very small deviations can result in large variances on the resulting worst-case response time.

## 5 Conclusion

We consider different models to access shared resources on multiprocessor systems and define schedules for tasks/superblocks on a processing element. The *sequential dedicated hybrid model* is shown to be schedulable as soon as any of the other models is schedulable. Experimental results demonstrate the superiority of the this with respect to worst case completion time. We conclude that resource sharing in multiprocessor systems should be designed according to the *sequential dedicated phases model*. In other words, accesses to the shared resource and computation should be separated carefully.

## References


1. M. Negrean, S. Schliecker, and R. Ernst. Response-time analysis of arbitrarily activated tasks in multiprocessor systems with shared resources. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pages 524–529, April 2009.
2. R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. Coscheduling of CPU and I/O transactions in COTS-based embedded systems. In *Proc. of the* $29^{th}$ *IEEE Real-Time System Symposium*, Dec 2008.
3. 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 '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 49–60, Washington, DC, USA, 2007. IEEE Computer Society.
4. S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst. Reliable performance analysis of a multicore multithreaded system-on-chip. In *CODES/ISSS '08: Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 161–166, New York, NY, USA, 2008. ACM.
5. Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, and Christian Ferdinand. Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 28(7):966–978, July 2009.