Timing Analysis of Multi-Mode Applications on AUTOSAR conform Multi-Core Systems

Mircea Negrean, Sebastian Klawitter, Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig, D-38106 Braunschweig, Germany
{negrean|ernst}@ida.ing.tu-bs.de

***Abstract*—Many real-time embedded systems execute multi-mode applications, i.e. applications that can change their functionality over time. With the advent of multi-core embedded architectures, the system design process requires appropriate support for accommodating multi-mode applications on multiple cores which share common resources. Various mode change and resource arbitration protocols, and corresponding timing analysis solutions were proposed for either multi-mode or multi-core real-time applications. However, no attention was given to multi-mode applications that share resources when executing on multi-core systems. In this paper, we address this subject in the context of automotive multi-core processors using AUTOSAR. We present an approach for safely handling shared resources across mode changes and provide a corresponding timing analysis method.**

I. Introduction

Modern hard-real time systems (e.g. multimedia smart devices, safety-critical avionic or automotive control systems) often run multi-mode applications, i.e. applications that have to change their functionality over time and execute in different operational modes. Driven by power constraints, cost-efficiency and performance requirements, multi-core architectures emerge as the prevalent platform in embedded real-time applications. This trend is most obvious in the automotive domain where multi-core solutions are provided by the semiconductor industry [1] and the AUTOSAR standard introduced support for partitioned multi-core OS [2].

In this context, the design process of embedded real-time systems faces new challenges generated by the need to accommodate multi-mode applications on multi-core architectures. Appropriate mechanisms that jointly handle the (i) mode management, (ii) multi-core scheduling and (iii) shared resource arbitration have to be defined in order to ensure a correct system functionality. Consequently, proper timing analysis methods for the prediction of timing behaviour of multi-mode multi-core applications are required.

Mode change protocols and *timing analysis approaches* dedicated to multi-mode single-processor [3]–[9] and multiprocessor real-time systems [10]–[12] were proposed. However, most related work assumes only independent tasks or neglects communication precedence relations between them.

Mode changes in distributed systems were addressed in [5], [7], [13]. Whereas [5] neglects communication precedence relations between tasks when reasoning about the tasks' timing during transition phases, in [7] the recurrent effect of a mode change was identified as significantly challenging the timing behaviour of distributed applications. The solution proposed in [13] captures the mode change recurrent effect and allows deriving mode change transition latencies.

978-3-9815370-0-0/DATE13/©2013 EDAA

The problem of sharing resources by multi-mode applications was studied in [3], [4], [6] but only for single-processor systems. For *asynchronous mode change protocols* [6], where *new mode tasks* may interfere with *old mode tasks*, it was shown that classic shared resource arbitration policies (e.g. the Priority Ceiling Protocol (PCP)), which are based on static task priorities and shared resource priority ceilings, are not valid anymore [4], [6] and counter the safe system functionality.

Several *arbitration protocols for inter-core shared resources* were proposed, e.g. the suspension-based Multiprocessor Priority Ceiling Protocol (MPCP) in [14] or its spin-based variant in [15]. Various *timing analysis solutions for multiprocessor systems with shared resources* were proposed e.g. in [15]–[18]. Note that we focus on partitioned multi-core setups, as global multiprocessor scheduling in safety-critical automotive applications has not (yet) found its way into practice.

The AUTOSAR standard introduced independent guidelines for either mode management [19], [20] or sharing resources in multi-core setups [2]. However, the complex setup consisting of multi-mode applications that share resources when executing on multi-core systems has not been considered so far.

In this paper, we fill this gap by providing (1) an approach for safely handling inter-core and intra-core shared resources across asynchronous mode changes and (2) a corresponding blocking- and response-time analysis approach. The contribution of this paper suits the next generation of automotive multi-core processors using AUTOSAR where applications will be scheduled using a partitioned fixed-priority preemptive scheduling, will share resources according to the AUTOSAR spinlock-based mechanism [2] and will be subject to mode management [19], [20].

Paper organization. After presenting the context of this work, in Sec. II we introduce a comprehensive system model that covers multi-mode and multi-core elements. In Sec. III we present our approach for handling shared resources across mode changes in multi-core systems. The corresponding timing analysis method is introduced in Sec. IV. We demonstrate the applicability of the proposed approach in Sec. V and draw conclusions in Sec. VI.

II. System Model

In this paper, we focus on multi-core architectures which consist of: (i) a set of m processor cores ($m \geq 2$), each being individually scheduled by a static priority preemptive (SPP) scheduler; (ii) local shared resources (LR), i.e. restricted to

individual cores, and global shared resources (GR), which can be accessed from each of the m cores. Shared resources are assumed to be objects that require serialized access. For the arbitration we consider: for LRs the Priority Ceiling Protocol (PCP) [14] and for GRs the AUTOSAR spinlock-based mechanism [2]; (iii) a static set of arbitrarily activated real-time tasks $T=\{\tau_1, \dots \tau_n\}$, where exclusive subsets of T are statically mapped to individual cores.

This system may execute in different operational modes specified by a finite set $M=\{M_1, M_2, \dots M_z\}$ ($z\in\mathbb{N}$). Each mode $M_i\in M$ is characterized by a different behaviour and is associated with a specific set of tasks (a subset of T) that are active in that mode. The possible transitions between two modes in M are specified by a finite set $\Phi=\{\Phi_{M1}^{M2}, \dots, \Phi_{M_y}^{M_z}\}$. A transition between two modes is initiated by a mode change request (MCR) triggered by the environment or by system internal requirements. The MCR is assumed as an global atomic event which may be triggered at any moment t_{MCR} during runtime. The possible overhead involved with the execution of the mode change protocols is not subject of this work. In order to exclude interference of multiple mode changes, a new MCR can be served only if the system is not executing a transition between two modes as a result of a previous MCR.

In this paper we address *asynchronous mode change protocols with periodicity* [6], [20]. This means, during a mode change we consider: *finished tasks* τ_{iF}, whose instances activated before t_{MCR} are allowed to finish after the MCR; *added tasks* τ_{iA}, which are activated with an offset ϕ any time later than the MCR (i.e. at $t_{MCR}+\phi$) and thus execute only in the new mode; and *unchanged tasks* τ_{iU}, which execute in both modes without any change in parameters.

Each instance of a task τ_i, called a job and denoted J_i, is activated by an event, which may be the result of a timer expiration, an external or internal interrupt, or of another task or bus communication being finished. We assume that the task graph which describes the functional and timing dependencies between tasks does not contain cyclic dependencies.

The activating events are modeled as arbitrary event streams [21], [22] using upper and lower event arrival functions $\eta^+(\Delta t)$ and $\eta^-(\Delta t)$. These specify the maximum and the minimum number of events that occur in the event stream during any time interval of length Δt. Inversely, event streams can be specified using the functions $\delta_i^+(n)$ and $\delta_i^-(n)$ that represent the largest and smallest time window in which n ($n\geq 2$) events can be observed in the stream.

Each job J_i of a task τ_i is further characterized by its static and unique priority given by the task numerical index (lower index means higher priority), a worst-case execution time C_i and a (relative) deadline D_i, which may be smaller, equal, or larger than the distance to the successive activation of the task. Jobs are executed in order, i.e. a new activated job will not execute before the previous job finishes.

During execution, each job can perform multiple non-nested accesses to LRs and GRs. Each of these accesses is considered a critical section guarded by a semaphore and protecting a LR or a GR. We differentiate between local critical sections (lcs) and global critical sections (gcs). The size of a lcs or of a gcs when it is accessed by jobs of a task τ_i are denoted ω_i^{LR} or ω_i^{GR}. With $\tilde{\eta}_i^{GRx}$ or $\tilde{\eta}_i^{LRx}$ we denote the load imposed by a job J_i on a global resource GRx or a local resource LRx.

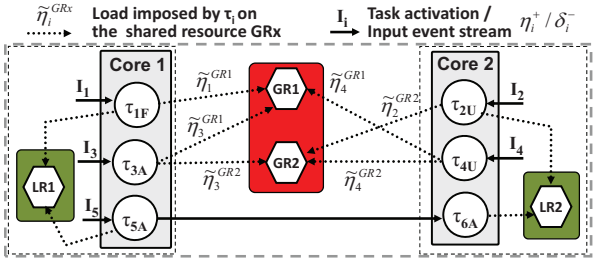

Fig. 1. Multi-mode multi-core system during a transition phase.

An example of a multi-mode multi-core system during a transition phase between two modes is illustrated in Fig. 1. We assume that a MCR imposes a mode change that consists in removing task τ_{1F} from Core1 and adding tasks τ_{3A}, τ_{5A} on Core1, and τ_{6A} on Core2. The unchanged tasks τ_{2U} and τ_{4U} execute independent of the mode change. I_1 to I_5 represent the event sources (given by the functions η^+ and δ^-) at the tasks input. The local and the global resources (i.e. $LR1$, $LR2$ and $GR1$, $GR2$) are accessed as indicated with the dashed lines.

III. Handling Shared Resources in Multi-Mode Multi-Core Systems using AUTOSAR 4.0

In order to handle the multi-mode behaviour of partitioned multi-core systems under AUTOSAR 4.0 the execution of the different types of tasks (τ_F, τ_A, and τ_U) has to be considered for the arbitration of both, LRs and GRs.

Regarding shared resources, the AUTOSAR 4.0 standard specifies a spinlock-based arbitration mechanism [2]. For the arbitration of LRs the AUTOSAR OS uses on the individual cores the classical PCP [14]. According to the PCP, each LR is allocated offline a static priority ceiling which is equal to the highest priority of any task which access that LR. At runtime, when a task locks a LR it inherits its associated priority ceiling.

Consequently, the obvious procedure for handling LRs in multi-mode systems is to allocate LRs multiple priority ceilings, one for each mode. Whereas this procedure is valid for individual modes, it can't be used during the transition phases controlled by asynchronous mode change protocols because [4], [6]: **(i)** if priority ceilings have to be raised but are adjusted too late, then an added task, released after the MCR, may inherit an old mode priority ceiling which is lower than its current priority. This violates the PCP, as priority ceilings must never be lower than the priority of any task using the resource; **(ii)** if priority ceilings have to be lowered but are adjusted too early then a finished task may inherit a new mode lower priority ceiling. Thus, activations of the finished tasks, executed after the MCR, could experience increased blocking in comparison to the activations executed before the MCR.

Both situations invalidate the existing blocking time analysis methods and counter the systems' timing behaviour.

In order to safely handle LRs, for each LR_k ($k\in\mathbb{N}$) a unique ceiling priority $CP(LR_k)$ has to be assigned such that it is valid for all operating modes in the set M.

For each local resource LR_k***, the only priority ceiling that is valid for all the operating modes is the so-called "ceiling of ceilings" [4], [6], that corresponds to the highest priority of any task*** τ_i ***accessing it in any mode*** $M_z\in M$:

$\forall LR_k\,(k\in\mathbb{N}), \forall M_z\in M\,(z\in\mathbb{N}), \forall\tau_i\in T$ and τ_i uses LR_k:

$$CP(LR_k)=\max(i) \qquad (1)$$

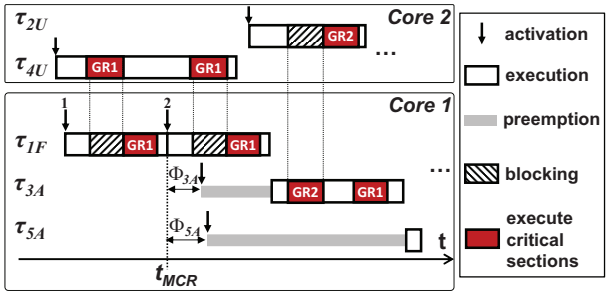

Fig. 2. Scheduling example for the system in Fig. 1.

For the arbitration of GRs the AUTOSAR 4.0 specifies the following: during execution, a task τ_i will actively wait (spin) if a requested GR is occupied by a remote task; during active waiting a task may be preempted by higher priority local tasks, but lower priority local tasks cannot start executing; if a task locks a GR it suspends all interrupts on his host core and thus it becomes non-preemptable; nested accesses to GRs are not allowed; if nesting is required, an explicit partial ordering of calls for GRs has to be predefined offline in order to avoid deadlocks and potential starvation situations.

As in AUTOSAR 4.0 conform multi-mode multi-core systems the arbitration of GRs is performed without using priority ceilings, there is no need of dynamically adjusting priorities across asynchronous mode changes. Therefore, there is no danger of violating the resource arbitration policy in systems with static tasks' priorities and implicitly blocking- and response-time analysis methods can be safely applied.

In Sec. IV we will introduce a timing analysis method for multi-mode applications scheduled on AUTOSAR 4.0 conform multi-core systems. Demonstrating that the blocking- and the response-times are bounded under all circumstances, we implicitly show that the procedure above for handling LRs and GRs across asynchronous mode changes is safe.

Before that, consider the scheduling example in Fig. 2 for the system in Fig. 1. During the transition phase initiated at t_{MCR}, the tasks τ_{3A} and τ_{5A} cannot start executing before task τ_{1F} finishes all activations corresponding to the old mode, i.e. initiated not later than t_{MCR}. According to the AUTOSAR specification, as τ_{1F} has higher priority than τ_{3A} and τ_{5A} even if τ_{1F} is blocked by the remote task τ_{4U}, the tasks τ_{3A} and τ_{5A} will not execute. Thus, lower priority new mode tasks cannot influence the execution of old mode higher priority tasks. Furthermore, as long as task τ_{1F} executes during the transition phase, none of the tasks τ_{2U} and τ_{4U} will experience blocking from τ_{3A}. If τ_{3A} would have higher priority than τ_{1F} then τ_{3A}'s execution would not wrongly be delayed by lower priority tasks. In other words, the AUTOSAR spinlock-based arbitration strategy inherently avoids the problems identified in case of using priority ceilings for LRs [4], [6].

IV. Timing Analysis for Multi-Mode Multi-Core Systems with Shared Resources

In order to derive the blocking- and the response-time analysis for multi-mode multi-core systems with shared resources, we rely on concepts from the real-time multiprocessor and multi-mode scheduling theory. More exactly, we rely on the *classic busy window* technique in [23] which was later used and extended in order to handle (i) multi-mode systems [6], [7], [13] or (ii) multi-core systems with shared resources [18].

The *busy window of a task* τ_i is generally defined as the time interval for which a resource executes only tasks of priority greater than or equal to the priority of task τ_i and during which the resource is never idle [23]. The maximum busy window of a task τ_i under SPP scheduling in partitioned multi-core systems with shared resources can be obtained by iteratively solving (2) [18].

$$w_i(q) = q \cdot C_i + B_i(w_i(q)) + \sum_{\forall \tau_j \in hp(i)} \eta_j^+(w_i(q)) \cdot C_j \quad (2)$$

where $w_i(q)$ is the maximum busy window of q activations of task τ_i where $q=1,\dots Q_i$ and $Q_i = \min\{q \geq 1 | w_i(q) < \delta_i^-(q+1)\}$, i.e. the iteration has to be continued as long as new activations of τ_i arrive before the previous finish; $B_i(w_i(q))$ is the maximum blocking time of τ_i in $w_i(q)$; $\eta_j^+(w_i(q)) \cdot C_j$ is the interference τ_i suffers due to the maximum workload of a higher priority job τ_j in $w_i(q)$.

The WCRT of a task τ_i is given by the largest response time of any of the q ($q=1,\dots Q_i$) task activations that lie within the busy window. The response time of the q-th activation of task τ_i is given by the difference between the window length $w_i(q)$ and the moment when this activation was initiated relative to the beginning of the busy interval. This is given by $\delta_i^-(q)$. The WCRT of any task τ_i is obtained with

$$R_i = \max_{q=1..Q_i} (w_i(q) - \delta_i^-(q)) \quad (3)$$

and the schedulability test consists in checking whether the condition $R_i \leq D_i$ holds for every task τ_i in the system.

However, when analyzing the schedulability of multi-mode systems under asynchronous mode change protocols one has to verify if tasks' deadlines are met in each individual mode and during every possible transition between two modes [6]–[8]. For each individual mode the classic analysis using eq. (2) and (3) above can be applied. But, in order to compute the WCRTs during each transition phase the *maximum transition busy window* (i.e. the maximum busy window during which a MCR occurs) has to be computed [6], [7]. Next, we extend the equations in [13] to calculate the transition busy windows and the WCRTs in multi-mode multi-core systems with shared resources.

A. Maximum Transition Busy Window in Multi-Core Systems

Our goal is to safely bound the timing behaviour of tasks in multi-mode multi-core systems with shared resources. Therefore, we compute the maximum transition busy window (abbr. MTBW) for each task by: **(I)** determining the worst-case scenario when the MCR shall occur such that it certainly leads to the worst-case execution during the transition phase and **(II)** by determining the maximum workload (denoted MW) of the different types of tasks (τ_F, τ_A, and τ_U) and their maximum blocking time in case of sharing resources.

(I) According to [6], [7] the **worst-case mode change scenario for a task τ_i is obtained when**: **(1)** t_{MCR} coincides with the activation instant of a finished higher priority task in $hp_F(i)$; **(2)** added tasks ($hp_A(i)$) are released with an offset ϕ after the initiation of the MCR and **(3)** unchanged higher priority local tasks in $hp_U(i)$ are assumed released simultaneously with τ_i, i.e in the classical critical instant.

These arguments, valid for uni-processor systems without shared resources, have to be investigated for multi-core setups.

For explanations we refer to the scheduling example in Fig. 3. Relying on the AUTOSAR arbitration policy introduced in Sec. III a task which has an outstanding request for a shared resource will actively wait for that resource without suspending. As illustrated in Fig. 3 each request of the tasks $\tau_{hpF}(i)$ and $\tau_{hpA}(i)$ on Core 2 for a GR is blocked by the remote task τ_{jU} on Core1. However, the lower priority tasks, e.g. τ_i, cannot start executing. This means that the blocking times due to the waiting for shared resources represent an extension of the tasks' core execution times. Therefore, the three arguments above (i.e. **(1)**,**(2)** and **(3)**) for constructing the worst-case mode change scenario also hold in case of multi-core systems using the AUTOSAR synchronization mechanism.

Regarding the identification of the worst-case mode change scenario, in case of arbitrary activated tasks there may be multiple higher priority finished tasks (tasks in $hp_F(i)$) and for each of these tasks there may be several possible activations (i.e. jobs) released at different moments in time, e.g. t_1 and t_2 in Fig. 3. Thus, in order to find the worst-case transition scenario one must identify all the time instances where the occurrence of the MCR should be assumed. The moments in time corresponding to the activations of the hp_F tasks are relative to the occurrence of the MCR at t_{MCR}. Let X_i be the set of all possible time intervals x_i (computed with the Algorithm 1 in [7]) relative to t_{MCR} which have to be investigated. The largest busy window obtained for any of the values x_i represents the *maximum busy window* of a task τ_i during which a MCR occurs.

(II) In order to compute the MTBW, we extend the busy window equation for multi-core systems (i.e. eq. (2)) to consider the maximum workload MW generated by the execution of unchanged, finished and added tasks and the maximum blocking time all these tasks can experience when waiting for the requested shared resources.

The maximum transition busy window in case of partitioned SPP scheduling of multi-mode multi-core systems with shared resources is obtained by iteratively solving (4).

$$w_i(q)=MW^i + B_i(w_i(q)) + \sum_{\forall \tau_F \in hp_F(i)} \eta^+_{\tau_F}(x_i)\cdot C_{\tau_F} + \sum_{\forall \tau_U \in hp_U(i)} \eta^+_{\tau_U}(w_i(q))\cdot C_{\tau_U} + \sum_{\forall \tau_A \in hp_A(i)} \eta^+_{\tau_A}(w_i(q)-x_i-\phi_{\tau_A})_0 \cdot C_{\tau_A} \quad (4)$$

with the maximum workload MW^i of the analyzed task τ_i:

$$MW^i = \begin{cases} q\cdot C_i; \text{if}(i==U)||(i==F) \\ \min(q, \eta_i^+(w_i(q)-x_i-\phi_i)_0)\cdot C_i; \text{if}(i==A) \end{cases} \quad (5)$$

The three sum terms in eq. (4) consider the MW due to the execution of higher priority finished, unchanged and added tasks. Activated but unfinished jobs of the finished tasks are assumed occurring in a time interval x_i starting before the initiation of the MCR at t_{MCR}. Added tasks are considered released with an offset ϕ_{τ_A} after t_{MCR}.

The clauses of (5) gives the MW of the analyzed task τ_i depending on its type. The second clause, i.e. if τ_i is an added tasks, indicates that, for large values of the offset ϕ_i, task τ_i does not contribute to the busy window $w_i(q)$. The function $\eta^+_{\tau_A}(w_i(q)-x_i-\phi_{\tau_A})_0$ represents a modified version of the original upper event arrival function $\eta^+(\Delta t)$ and returns 0 if $w_i(q)-x_i-\phi_i<0$.

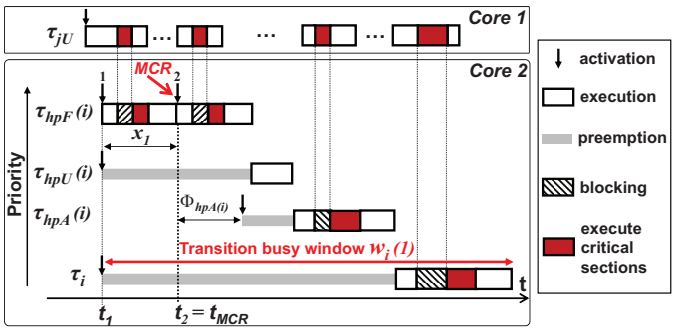

Fig. 3. Scheduling example for a task τ_i during a mode change where MCR coincides with the 2nd activation of the higher priority finished task.

Regarding the tasks' blocking times these also depend on the system's multi-mode behaviour. Therefore, the factor $B_i(w_i(q))$ in eq. (4) has to be derived by considering the execution of the different types of tasks on all the system's cores. As can be seen in Fig. 3 when analysing a task τ_i its execution is delayed not only by its direct blocking times but also by the blocking times of the higher priority local tasks. For any analysed task τ_i all these blocking times are captured by the analysis presented in Sec. IV-B and are part of the factor $B_i(w_i(q))$. Furthermore, note that Algorithm 1 in [7], used for computing all possible values x_i, essentially requires the computation of the maximum busy window with a MW of finished and unchanged tasks, i.e. the longest busy window within the old mode. Thus, for the analysis of multi-mode multi-core systems the equations for computing the values x_i have to be extended with the blocking time factor $B_i(w_i(q))$.

B. Blocking Time Analysis in Multi-Mode Multi-Core Systems

In this section, we introduce a blocking time analysis for arbitrarily activated tasks that share resources, arbitrated by the AUTOSAR mechanisms, in multi-mode multi-core setups. The following terms are used later in the blocking time factors:

- n_i^G is the maximum number of global critical sections that each job J_i of a task τ_i executes before its completion.
- $\tilde{\eta}_i^+(w_i(q))$ is the *Shared Resource Request Bound*[1] which represents the maximum number of requests that may be issued by a task τ_i to a shared resource within the investigated time interval $w_i(q)$.
- ω_i^{LR} and ω_i^{GR} represent the maximum duration of a lcs and of a gcs when it is accessed by jobs of a task τ_i.
- $lpl(i)$ and $hpl(i)$ are the sets of tasks mapped on the same core as τ_i and have lower and higher priority than τ_i.
- $lpr(i)$ and $hpr(i)$ are the sets of tasks mapped on remote cores and have lower and higher priority than τ_i.
- $GS_{i,j}$ represents the set of global semaphores that will be locked by jobs of both tasks τ_i and τ_j.
- The set of tasks which are elements of $lpr(i)$ and access elements of $GS_{i,j}$ is denoted with $\theta_{i,j}$.
- Similar the set of tasks which are elements of $hpr(i)$ and access elements of $GS_{i,j}$ is denoted with $\Theta_{i,j}$.
- Jobs of the tasks in $\theta_{i,j}$ and $\Theta_{i,j}$ are jobs which directly block jobs of task τ_i.

[1]The derivation of $\tilde{\eta}_i^+(\Delta t)$ can be made by investigating the tasks internal control flow [24] or by relying on the event model concept used to model task activations to also capture the resource traffic [18].

Remember that the *SharedResourceRequestBound* function and the sets of considered tasks have to capture the specific type (τ_U,τ_F,τ_A) of tasks that are subject of blocking. Based on the AUTOSAR specification [2] and on the terms introduced above the blocking time of a job J_i in a partitioned multi-mode multi-core system consists of the following factors:

1. Local blocking time. A job J_i of a task τ_i may be blocked only once by a job J_j of a lower priority local task $\tau_j \in lpl(i)$. As nesting is not allowed, J_j can either execute a lcs or a gcs. Thus, the local blocking time of a job J_i is bounded by:

$$B_{i1}(w_i(q)) = \max(\omega_j^{LR}, \omega_j^{GR}) \tag{6}$$

$$\text{with} \begin{cases} \tau_j \in lpl_U(i) \bigcup lpl_F(i); \text{if}(i==F) \\ \tau_j \in lpl(i); \text{if}(i==U)||(i==A) \end{cases}$$

The first clause above captures the case where τ_i is a finished task. In this case lpl_A tasks cannot start and queue up for any shared resource and thus these cannot block τ_i. The second clause captures the case where τ_i is an unchanged or an added task, that can be blocked by one previously released task τ_j of any type, i.e. $\tau_j \in lpl(i) = lpl_U(i) \bigcup lpl_F(i) \bigcup lpl_A(i)$.

2. Direct blocking time. Each time a job J_i tries to access a gcs, this can be currently locked by a lower priority remote job of a task $\tau_j \in lpr(i)$. Thus, the blocking time due to lpr tasks which share the same GR with J_i is given by the longest gcs of tasks in the set $\theta_{i,j}$:

$$B_{i2}(w_i(q)) = q \cdot n_i^G \cdot \max_{\forall \tau_j \in \theta_{i,j}} (\omega_j^{GR}) \tag{7}$$

Each job J_i can also be blocked by hpr jobs that request the same GR as J_i (i.e. by jobs of tasks in the set $\Theta_{i,j}$). As on the remote cores there can be tasks of different types, the load $\tilde{\eta}_j^+$ imposed by them on the GRs accessed by τ_i during the transition busy window $w_i(q)$ has to consider their types:

$\forall \tau_j \in \Theta_{i,j}$:

$$\tilde{\eta}_j^+ = \begin{cases} \tilde{\eta}_j^+(x_j); \text{if} \tau_j \in hpr_F(i) \bigcap \Theta_{i,j} \\ \tilde{\eta}_j^+(w_i(q)); \text{if} \tau_j \in hpr_U(i) \bigcap \Theta_{i,j} \\ \tilde{\eta}_j^+(w_i(q) - x_j - \phi_j); \text{if} \tau_j \in hpr_A(i) \bigcap \Theta_{i,j} \end{cases} \tag{8}$$

As opposed to lpr jobs, hpr jobs may be served multiple times before the J_i will be able to lock the GR. Thus, the direct blocking time due to hpr tasks is given by:

$\forall \tau_j \in \Theta_{i,j}$ and $\tilde{\eta}_j^+$ given by (8):

$$B_{i3}(w_i(q)) = \sum \tilde{\eta}_j^+ \cdot \omega_j^{GR} \tag{9}$$

As can be observed in eq. (8) and (9) the blocking time of a task τ_i, investigated for one time interval x_i relative to t_{MCR}, depends on the time intervals x_j relative to t_{MCR} that have to be investigated for other finished or added tasks τ_j on other cores. This dependency can be handled by integrating the blocking-time analysis into a compositional system-level analysis procedure [21], [22] (see Sec. IV-C).

3. Busy-waiting of higher priority local tasks. A job J_i cannot start executing on its host core as long as hpl tasks are actively waiting for the required GRs. As hpl tasks do not suspend when waiting for GRs, their direct blocking times caused by the remote tasks represent also a blocking of J_i:

$$B_{i4}(w_i(q)) = \sum_{\forall \tau_j \in hpl(i)} (B_{j2}(w_i(q)) + B_{j3}(w_i(q))) \tag{10}$$

$B_{j2}(w_i(q))$ and $B_{j3}(w_i(q))$ in (10) are calculated with (7) and (9) for all tasks $\tau_j \in hpl(i)$.

The worst-case blocking time $B_i(w_i(q))$ that a job of a task τ_i can encounter in a time window $w_i(q)$ is given by the sum of the 4 blocking factors above, i.e. (6), (7), (9) and (10).

C. Multi-Mode Multi-Core Timing Analysis Approach

The WCRTs are computed by integrating the blocking times in Sec. IV-B in the MTBW computation with eq. (4). A solution for eq. (4) can be computed iteratively, because all components (i.e. number of considered tasks activations η_j^+ and therewith the maximum workload of the tasks, and the load imposed on the shared resources $\tilde{\eta}_j^+$) grow monotonically with respect to the window size [18]. The WCRT of a task τ_i is given by the largest response time R_i of any of the q activations (q=1..Q_i, Q_i=$\min\{q \geq 1 | w_i(q) < \delta_i^-(q+1)\}$) that lie within the MTBW $w_i(q)$, i.e.

$$R_i = \begin{cases} \max(w_i(q) - \delta_i^-(q)); \text{if}(i==U)||(i==F) \\ \max(0, w_i(q) - x_i - \phi_i - \delta_i^-(q)); \text{if}(i==A) \end{cases} \tag{11}$$

The clauses in (11) states that depending on the task's type, R_i is obtained by subtracting from $w_i(q)$ the distance between the start of the transition busy window and the activation instant of the q-th job. If τ_i is an added task which is not activated within the transition busy window, R_i is 0.

As can be observed from eq. (4), (8), (9) the busy window w_i and therewith the response time R_i of a task depend on the load $\tilde{\eta}_j^+$ imposed on the shared resources by tasks on other cores and potentially by their worst-case time interval x_j where the MCR shall occur. This dependency can be solved by embedding the response- and the blocking-time analysis into a compositional analysis methodology such as [21] or [22].

Thus, the system-level analysis is an iterative process which performs for each task on each core **(i)** the above response time analysis which includes the computation of the MTBWs, **(ii)** the analysis of all possible time intervals x_i where the MCR shall occur in order to lead to the worst-case behaviour during the transition phase and **(iii)** the blocking time analysis which requires the investigation of all possible time intervals x_j of other tasks on other cores, until the definite event models have been found. After convergence, the schedulability test consists of checking whether $R_i \leq D_i$ holds for every task τ_i.

V. Experiments

To demonstrate the applicability and the benefits of the proposed approach we compare it to the currently available design procedure for AUTOSAR multi-core systems. The current design practice, which is not multi-mode aware, can safely handle the system in Fig. 1 only by assuming that all tasks are always running on the two cores, i.e. by modelling all tasks as unchanged, not only in the individual modes but also during the transition phase.

Hence, for the transition phase of the system in Fig. 1, we apply both, (a) the classic response-time analysis method for the case where all tasks are modelled as unchanged and (b) our approach (in Sec. III and IV) which is able to handle the multi-mode behaviour of the multi-core system.

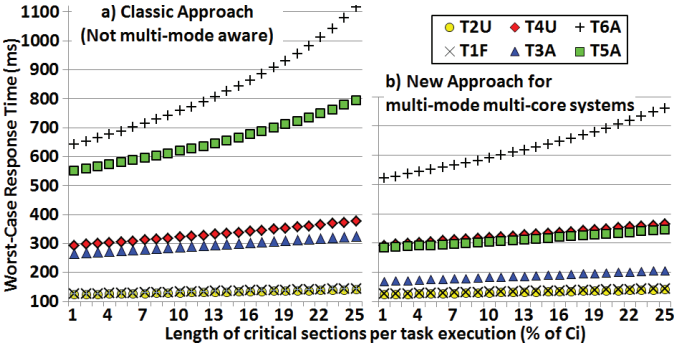

Fig. 4. Tasks' WCRTs depending on the critical sections length: a) current design practice; b) our approach for multi-mode multi-core systems.

For the evaluation we randomly generated test cases until we got 1000 schedulable configurations of the system in Fig. 1. The test cases were generated such that: the load on each core was 50%; the load on a core was randomly distributed among tasks; the tasks' periods P_i were generated randomly between 100 and 1000ms; the tasks' execution times C_i were computed based on the tasks' periods and loads. Each task was randomly assigned an input jitter from the interval $[0, 2 \cdot P_i]$, i.e. we generated a burst of maximum 3 activations. Each task performs two requests for each LR and GR it uses during C_i. The total length of the critical sections per C_i was equally split among the number of requests. Based on the number and on the size of the critical sections the distance between every two requests d_{srr} was modelled such that critical sections are equally spread across the C_i. Thus, the load imposed on the shared resources was calculated with $\tilde{\eta}_i^+(\Delta t) = \lceil \Delta t / d_{srr} \rceil$.

For each test case, the total length of the task's critical sections was varied from 1% to 25% of the C_i. Fig. 4 a) and b) depict the tasks' WCRTs depending on the critical sections' length. For each task the average WCRT over the 1000 setups per critical section length is given.

As expected, independent of the design approach, increasing the size of the critical sections led to increased blocking times and therewith to increased response-times. However, when comparing the results of the two approaches, one can see that our proposed approach greatly takes advantage of its ability of handling the different types of tasks across mode changes. Whereas for the higher priority tasks τ_{1F} and τ_{2U}, there is no difference, as the AUTOSAR spinlock-based arbitration always favours them, for the other tasks, the response-times computed with our approach are in average 30.5% lower. More exactly, there is an average improvement of 1.5% for task τ_{4U}, of 42% for task τ_{3A}, 53% for task τ_{5A} and 25.5% for task τ_{6A}.

The tests (non-optimized code) were performed on an Intel Core i5 M460 2.53GHz CPU, 4GB RAM, 64bit Windows and took in average 358ms for each analyzed configuration.

VI. Conclusion

In this paper, we presented an approach for the design and analysis of multi-mode applications that share resources in multi-core systems. We focused on the next generation automotive multi-core processors using AUTOSAR, for which there was no support for jointly addressing the problems of (i) mode management, (ii) multi-core scheduling and (iii) shared resource arbitration. The method we proposed allows to safely handle local and global shared resources across asynchronous mode changes in AUTOSAR systems. The corresponding timing analysis approach provides timing guarantees for multi-mode applications consisting of tasks with arbitrary activation patterns. The applicability of the presented solution and its benefits against existing approaches was demonstrated with the analysis of a set of pseudo-randomly generated test cases.

References

[1] Infineon, "AURIX Safety joins Performance," [Online]. Available http://www.infineon.com/aurix, July 2012.

[2] AUTOSAR GbR, "Specification of Operating System R4.0 v5.0.0," [Online]. Available http://www.autosar.org/, November 2011.

[3] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode Change Protocols for Priority-Driven Preemptive Scheduling," *Real-Time Systems*, vol. 1, pp. 243–264, 1989.

[4] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode Changes in Priority Pre-emptively Scheduled Systems," in *Proc. of the Real-Time Systems Symposium*, 1992, pp. 100–109.

[5] P. Pedro, "Schedulability of Mode Changes In Flexible Real-Time Distributed Systems," Ph.D. dissertation, University of York, Sep. 1999.

[6] J. Real and A. Crespo, "Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, March 2004.

[7] R. Henia and R. Ernst, "Scenario Aware Analysis for Complex Event Models and Distributed Systems," in *Proc. 28th IEEE RTSS*, Dec. 2007.

[8] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling," in *Design, Automation and Test in Europe (DATE)*, April 2009, pp. 99–104.

[9] L. T. Phan, I. Lee, and O. Sokolsky, "Compositional Analysis of Multi-Mode Systems," in *Proc. of 22nd ECRTS*, July 2010.

[10] V. Nelis, J. Goossens, and B. Andersson, "Two Protocols for Scheduling Multi-mode Real-Time Systems upon Identical Multiprocessor Platforms," in *Proc. of 21st ECRTS*, July 2009, pp. 151–160.

[11] P. M. Yomsi, V. Nélis, and J. Goossens, "Scheduling multi-mode real-time systems upon uniform multiprocessor platforms," in *IEEE Conf. on ETFA*, Sept. 2010, pp. 1–8.

[12] M. Negrean, S. Schliecker, and R. Ernst, "Mastering Timing Challenges for the Design of Multi-Mode Applications on Multi-Core Real-Time Embedded Systems," in *6th Int. Congress on ERTS*^{2}, February 2012.

[13] M. Negrean, M. Neukirchner, S. Stein, S. Schliecker, and R. Ernst, "Bounding mode change transition latencies for multi-mode real-time distributed applications," in *IEEE Conf. on ETFA*, Sept. 2011, pp. 1–10.

[14] R. Rajkumar, *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publ. Norwell, MA, USA, 1991.

[15] K. Lakshmanan, D. d. Niz, and R. Rajkumar, "Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors," in *Proc. of 30th IEEE RTSS*, 2009, pp. 469–478.

[16] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proc. 22nd IEEE RTSS*, Dec. 2001, pp. 193–202.

[17] B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?" in *Proc. of IEEE RTAS*, April 2008.

[18] S. Schliecker, M. Negrean, and R. Ernst, "Response Time Analysis on Multicore ECUs with Shared Resources," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 4, pp. 402–413, November 2009.

[19] AUTOSAR GbR, "Guide to Modemanagement R4.0 v1.0.0," [Online]. Available http://www.autosar.org/, October 2011.

[20] ——, "Specification of RTE R4.0 v3.2.0," [Online]. Available http://www.autosar.org/, November 2011.

[21] S. Chakraborty, S. Künzli, and L. Thiele, "A General Framework for Analysing System Properties in Platform-based Embedded System Designs," in *Design, Automation and Test in Europe (DATE)*, March 2003, pp. 190–195.

[22] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - The SymTA/S Approach," *IEE Proc. Comp. and Digital Tech.*, vol. 152, no. 2, pp. 148–166, Mar. 2005.

[23] K. W. Tindell, A. Burns, and A. J. Wellings, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks," *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.

[24] K. Albers, F. Bodmann, and F. Slomka, "Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems," *Proc. of 18th ECRTS*, 2006.