Deconstructing Bus Access Control Policies for Real-Time Multicores

Javier Jalle[†,*], Jaume Abella[†], Eduardo Quiñones[†], Luca Fossati[⋆], Marco Zulianello[⋆], Francisco J. Cazorla[†,‡]

[†]Barcelona Supercomputing Center, Spain
[*]Universitat Politècnica de Catalunya, Spain
[⋆]European Space Agency, Netherlands
[‡]Spanish National Research Council (IIIA-CSIC), Spain

***Abstract*—Multicores may satisfy the growing performance requirements of critical Real-Time systems which has made industry to consider them for future real-time systems. In a multicore, the bus contention-control policy plays a key role in system's performance and the tightness of the Worst-Case Execution Time (WCET) estimates.**

In this paper we develop analytical models of the contention that requests from different tasks running in different cores suffer for the two most-used contention control policies: Time-Division Multiple Access (TDMA) and Interference-Aware Bus Arbiter (IABA), which allows us to compare them. We further show the benefits of having such models for real-time system designers and chip providers.

Our results show that WCET estimates obtained with TDMA are slightly (2%) tighter than those obtained with IABA, at the cost of knowing the exact cycle at which every access of every task accesses the bus. However, average performance is 10% worse with TDMA than with IABA. Overall, IABA is the most appealing contention-control policy since it allows achieving tight WCET estimates and high average performance with little burden for the user.

I. Introduction

Critical Real-Time Embedded System (CRTES) market is experiencing an unprecedented growth [5][8] with rising demands for greater performance (computing power). High performance requirements can be met by designing more complex processors, e.g. with out-of-order execution and higher clock frequency. However, applying those designs to CRTES design is difficult, because (1) they could introduce timing anomalies [15] due to their non-deterministic runtime behavior and (2) they have high energy requirements that affect the low-power constraints of embedded systems.

Multicores maintain a simple processor design, have a good performance-per-watt ratio and enable co-hosting of mixed-criticality workloads (e.g. with high data-processing demand and stringent time constraints), which is of paramount importance in the embedded system market because hardware utilization is maximized while cost, size, weight and power requirements are reduced. However, the use of multicores in CRTES is not straightforward.

On the one hand, any application must be prevented from corrupting the state of other applications, paying special attention to preventing low-criticality applications from affecting high-criticality ones. This can be accomplished through software isolation [11].

On the other hand, CRTES require guarantees on the timing correctness of the system, which is obtained by estimating the WCET (*worst case execution time*) for each task and properly scheduling them. However, timing behavior is much harder to analyze for multicore processors than for single-core ones, mainly due to *inter-task interferences*. Inter-task interferences appear when several tasks in different cores try to access a shared hardware resource at the same time – e.g. a shared bus or a shared cache –, thus creating contention, potentially affecting the execution time of running tasks. As a result, providing a meaningful timing analysis becomes difficult, since the execution time of a task is affected by the other tasks running simultaneously in the processor.

One of the most critical shared resources in multicore processors is the interconnection network. There is a common belief that the bandwidth necessary for future applications can only be provided by deploying networks like meshes with complex routers. However, several studies show that hierarchical bus configurations scale quite easily to large systems and provide a good area-performance trade-off, while retaining many of the advantageous features of simpler bus arrangements [21]. In the same line, other studies show that bus-based networks can significantly lower energy consumption and simplify network protocol design and verification, with no loss in performance [23]. For instance, the Advanced Microcontroller Bus Architecture (AMBA) [1] is used not only in microcontroller devices but also on a range of ASIC and SoC parts with real-time capabilities.

The contention among several cores attempting to access the bus simultaneously, generates inter-task interferences in the bus that are handled by the arbitration policy. The choice of the policy affects the whole system because the time a bus request takes to be completed, depends on the amount of time that the request waits to be granted access by the arbitration policy, which intrinsically depends on the other running tasks (or system's workload). In order to be able to provide WCET estimates, the effect of inter-task interferences has to be known or, at least, upper bounded for every possible workload (ideally this bound is independent of the workload) and introduce as little pessimism in the WCET estimate as possible. If that upper-bound depends on the set of running tasks, the WCET analysis has to consider all possible interactions within every workload, thus challenging time composability[1], which significantly complicates the analysis and integration of the system. Thus, using an appropriate arbitration policy will help (1) simplifying WCET analysis for multicore systems and (2) providing tight WCET estimates.

To our knowledge, two main policies have been proposed to deal with *inter-task interferences on on-chip buses* that satisfy these requirements: Time-Division Multiple Access (TDMA) and Interference-Aware Bus Arbiter (IABA) [16].

TDMA applies time sharing between the requests of the different contenders. Time is divided into *windows*, in each of which a contender is assigned a *slot*. When a request of a given contender becomes ready in that contender's slot, the request gets immediate access to the bus. If the request becomes ready out of the contender's slot, it waits until the contender's next slot. IABA, instead of time-sharing the access to the bus, allows tasks to contend for the bus using a given access policy such as round-robin, and bounds the delay that a request can suffer due to inter-task interferences under that access policy. This delay is taken into account when computing WCET estimates for each task, that covers the maximum effect that inter-task interferences can have on that task, thus providing safe WCET estimates.

[1]Time composability stands for the property of allowing tasks to be analyzed in isolation obtaining WCET estimates that hold regardless of the workload in which the task analyzed is executed.

978-1-4799-0658-1/13/$31.00

The main contributions of this paper are the following: (1) We develop analytical models of the contention delay that each of those techniques introduce, which potentially affects the WCET of applications. (2) While both techniques have been widely analyzed in the past [20][13][16][26], no study compares them. In this paper we cover that gap by providing the first comparison between TDMA and IABA. Our study provides means to choose between TDMA and IABA in early stages of the hardware design, which is attractive to chip vendors given that buses are widely used nowadays in CRTES, and if both policies are available in the processor, our study guides software designers to choose the policy that better fits their needs.

For comparison purposes, we use several key metrics in the design of CRTES: WCET estimates using a commercial WCET analysis tool, time composability, average performance, prioritization capabilities and amount of changes required to the WCET analysis tool. Our results show that IABA represents a better choice for real-time multicores: IABA requires no changes in the WCET analysis tool, provides higher average performance and better WCET estimates in those scenarios in which the exact cycle when requests access the bus cannot be determined by the WCET analysis tool. TDMA, instead, slightly outperforms IABA only if the exact cycle in which each request accesses the bus can be determined, which is hard – if at all feasible – to obtain in general.

Given that several real processors deployed by real-time industries use buses as the main communication channel [12][3][2], we believe that our study provides valuable information for real-time system designers and chip vendors mainly in the early design stages to choose which policy better fits their needs (more details in Section IV).

The rest of our paper is organized as follows: Section II describes both TDMA and IABA, which are later compared in Section III. Section IV describes how industry can use the results of this work. Finally, Section V presents the main conclusions of this study.

II. On-chip Bus Access Policies

When more than one Hard Real-time Task (HRT) run simultaneously on the processor, it may happen that two or more requests from different HRTs attempt to access the bus at the same time. In this case, the arbitration policy decides which HRT is granted access to the bus and which one has to wait. Hence, the HRT that is granted access to the bus potentially delays other HRTs. Under some arbitration policies, there may not be an upper bound to the time one HRT can delay the others to access the bus. For instance, let us assume two HRTs, HRT_1 and HRT_2, so that HRT_1 has higher priority than HRT_2 and both tasks attempt to access the bus simultaneously. In this situation, HRT_2 is stalled until HRT_1 finishes. However, if before the first request from HRT_1 finishes, another request from HRT_1 becomes ready, HRT_2 will also wait for the second request to finish as well. Thus, the upper bound that HRT_2 suffers due to inter-task interferences depends on HRT_1. Even though this upper bound can be computed knowing HRT_1 sequence of accesses, it can be too long/pessimistic to be useful. Moreover, it breaks time composability, the property according to which the WCET estimate for a task τ_i can be computed in isolation and it is not affected by the other tasks that may run concurrently with τ_i (see Section III-E).

The effect of inter-task interferences, which directly depends on the arbitration policy and indirectly on the set of running tasks (workload), has to be known or at least upper-bounded for every request in order to be able to provide safe WCET estimates. If the upper-bound of that effect depends on the other tasks running concurrently (as it is the case with the priority arbitration policy), time composability is broken (see Section III-E). Hence, WCET analysis becomes significantly more complex – if at all attainable – because it must consider all the possible interactions within every possible workload.

TDMA and IABA arbitration policies, allow to upper-bound the effect of inter-task interferences regardless of the workload, for every request of a given task and provide tight WCET estimates. Both policies simplify the WCET analysis and allow using existing WCET analysis tools for single-core processors to analyze the timing of multicores.

We base our study in a multicore architecture in which each core has private instruction and data caches, which is the common practice in current high-performance and real-time embedded processor designs [3][12]. A bus connects the cores to the shared L2 cache. Hence, a task (core) sends a request to the bus on (1) every L1 data cache load miss, (2) L1 instruction cache miss and (3) store operation since we model a write-through data cache. Memory operations that miss in L2 cache are sent to the memory controller that is connected to the L2 cache. When a core is ready to send a request to the bus at the beginning of a given cycle n, it sends a signal to the bus arbiter and if the core is granted access, it accesses the bus in the next cycle $n + 1$. Instead, if the core is not granted access, its request has to wait. The delay a request suffers to get access to the bus is called *Bus Inter-task Delay* (BID), being the total time (t_i) of a bus request i to reach the appropriate destination, $t_i = BID_i + r_i$, where r_i is the actual time that the request owns the bus, or request latency, which depends on the amount of data to transfer and the bus characteristics (e.g., width, latency). We will focus on BID, since r_i is independent of inter-task interferences, so we assume that r_i is known, or at least, upper-bounded, as it would be in the single-core case.

A. TDMA

TDMA has traditionally been used to arbitrate communications in distributed embedded systems (like the Time-Triggered Architecture [14]). Recently, several studies propose TDMA to be used for communications on MPSoCs. In particular in [20][13] TDMA is used as the bus arbitration policy in a bus-connected multicore processor.

TDMA splits time into windows of size w cycles. Every window is divided into slots of size s and each contender to the bus (processor cores in our case) is assigned a slot. During a given slot, only its owner can send requests. If that contender has no requests to send, the bus will remain idle for that slot, even if there are requests from other contenders trying to access the bus (non-work-conserving). Slots can have fixed or variable length. Several approaches have been proposed to optimize TDMA slot size when TDMA is used in distributed systems [24] or to connect multiple cores [20] using variable slot length. In this paper, TDMA uses equally-sized slots and the TDMA time window configuration chosen is used for all applications, not to affect systems time composability. We elaborate more on this point in Section III-E.

Figure 1 shows a scenario with $N_c = 4$ cores, each having a TDMA slot of length $s = 4$ processor cycles and a window length $w = N_c \times s = 16$. A task on core 1 can only access the bus in the second slot ($n \in [4, 7]$). If the task sends a request when its slot has just elapsed, it has to wait for the rest of other core's slots to finish, i.e., $(N_c - 1) \times s$ cycles, even if there is no other request

	C0				C1				C2				C3			
cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BID	4	3	2	1	0	0	0	13	12	11	10	9	8	7	6	5
Case	A				B			C	D							

Fig. 1. BID figures for a request of core 1 arriving in different cycles under the TDMA setup: $w = 16$, $N_c = 4$, $s = 4$ and $r = 2$

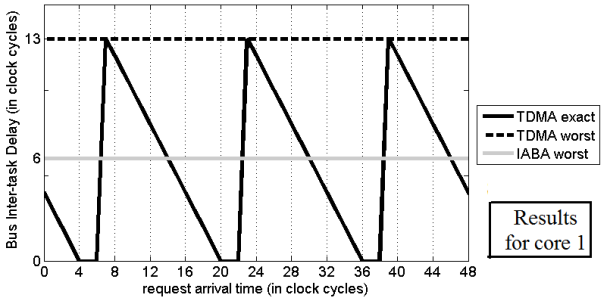


Fig. 2. TDMA and IABA results with $w = 16$, $s = 4$, $N_c = 4$ and $r = 2$.

pending. If each request has a latency of $r = 2$ cycles (obviously, it is always required that the request fits inside the slot, i.e., $s \geq r$) and the request of a task arrives in the last $r - 1$ cycles of the task's slot, it cannot be granted access to the bus since it would span into the slot of the next core. Indeed, this is the worst situation in terms of BID, which is given by $BID^{TDMA}_{worst} = (N_c - 1)s + (r - 1)$, i.e., 13 in the scenario drawn in Figure 1. BID^{TDMA}_{worst} could be taken as a safe upper bound of the inter-task interferences, though it would be often pessimistic (i.e. overestimated). In reality, the exact effect of inter-task interferences that a given request from a task suffers, depends on the cycle in which that bus request arrives with respect to that task's TDMA slot. Note that in both cases (exact and worst case), the computation of the effect of inter-task interferences, does not depend on the other contenders, which allows to derive WCET estimates in isolation for each task, regardless of the workload.

Let's assume that a bus request from thread/core c (where $0 \leq c \leq N_c - 1$), arrives at the absolute cycle n, measured in processor cycles. The slot $i \in [0, N_c - 1]$, in which the request arrives is given by $i = \lfloor n/s \rfloor \bmod N_c$. n translates into relative cycle $rc \in [0, s-1]$ within slot i as $rc = n \bmod s$. If i matches the core id, c then the request has arrived in its core slot. There are four different scenarios for the computation of the effect of inter-task interferences depending on the relation between the arrival slot i, the sending core c, and whether the request fits in the current slot or not. Table I shows these four cases (A-D). Cases A and D correspond to issuing the request before and after its own slot respectively; in case B the request is issued in its slot and has enough cycles left to proceed, and in case C the request is issued in the correct slot but without enough cycles left. Only in case B the waiting time is 0 because the request arrives in its corresponding slot and has enough time to be sent. The time the request has to wait, in cycles, to get access to the bus depends on the relative cycle, rc, in which the request arrives, so BID^{TDMA}_{exact} is:

$$BID^{TDMA}_{exact} \begin{cases} (c - i - 1) \cdot s + (s - rc) & \text{case A} \\ 0 & \text{case B} \\ (N_c - i + c - 1) \cdot s + (s - rc) & \text{cases C,D} \end{cases} \quad (1)$$

Equation (1) matches BID values in Figure 1, in the scenario explained before. Figure 2 presents a graphical view of the different TDMA BID values. We observe that BID^{TDMA}_{exact} has sawtooth behavior. Also it can be seen that $BID^{TDMA}_{worst} \geq BID^{TDMA}_{exact}$ for any given cycle, which shows the pessimism when obtaining WCET estimates using BID^{TDMA}_{worst} instead of using BID^{TDMA}_{exact}.

TABLE I. ARRIVAL TIMES OF A BUS REQUEST

Case	Expression
A	$i < c$
B	$i = c$ and $(s - rc) \geq r$
C	$i = c$ and $(s - rc) < r$
D	$i > c$

B. IABA

IABA is devised to work on top of an existing arbitration policy. In this paper, we focus on round-robin because it is a commonly used and simple policy, but the same analysis can be applied to any arbitration policy that allows to upper bound the effect of inter-task interferences. Conceptually, in round-robin all contenders have different priorities, and those priorities are rotated across contenders on every arbitration. Therefore, one core will have the lowest priority in a particular arbitration, the second lowest in the next arbitration, the third lowest in the next arbitration, and so on and so forth until it gets the highest priority after $N_c - 1$ arbitrations. Then, in the next arbitration it gets the lowest priority again. In this way, if one contender has no pending requests, the next contender will be served, being the bus idle only when there are no requests (*work-conserving*). Under round-robin, the maximum delay a request can suffer due to inter-task interferences (contention) occurs when all the other contenders are trying to access the bus and have higher priority. This maximum delay is called BID^{IABA}_{worst} (which is called Upper-Bound Delay (UBD) in [16]). BID^{IABA}_{worst} is bounded by the maximum number of contenders that can send a request at the same time. The latter is, in turn, bounded by the number of cores (N_c). Overall, BID^{IABA}_{worst} is computed as follows, where r is the request latency:

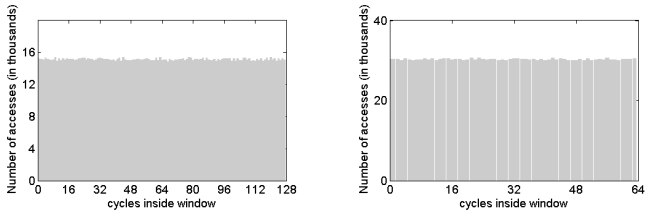
$$BID^{IABA}_{worst} = (N_c - 1) \cdot r \quad (2)$$

IABA always assumes BID^{IABA}_{worst} for every access to the bus, in order to be able to perform the WCET analysis of any task in isolation, regardless of the workload (in [9], they reduce the value of BID^{IABA}_{worst} based on the possible workloads, however this breaks the time composability property). This assumption introduces pessimism in the WCET estimate but significantly simplifies the analysis.

Let's assume the same configuration used in Figure 1 with N_c=4 and r=2. Further assume that we use round-robin policy. Under this setup, the maximum inter-task delay a request can suffer is 6 cycles, i.e. $BID^{IABA}_{worst} = 6$, which is the delay that corresponds to the accesses of the other 3 cores. BID^{IABA}_{worst} is also shown in Figure 2 for this setup. As it can be observed, BID^{IABA}_{worst} is constant regardless of the cycle in which the request arrives, since IABA assumes the worst case for each request. Hence, with IABA the maximum delay that a request will suffer due to bus interferences depends on the number of HRTs that are going to be executed simultaneously in the multicore processor, which is bounded by the number of cores N_c, thus eliminating the dependence on the arrival cycle of requests.

C. Analytical comparison of TDMA and IABA

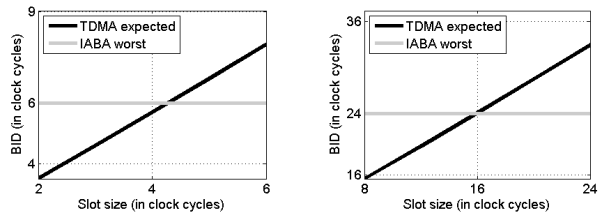
We note that in order to minimize the worst-case value, the best choice for TDMA is to use the smallest possible slot size, which is $s = r$, the minimum slot size. In that case $BID^{IABA}_{worst} = (N_c -$



(a) 128-cycle window (b) 64-cycle window

(c) 32-cycle window (d) 16-cycle window

Fig. 3. Memory accesses histogram inside a TDMA for several window sizes



(a) $N_c = 4$, $r = 2$ (b) $N_c = 4$, $r = 8$

(c) $N_c = 4$, $r = 16$ (d) $N_c = 8$, $r = 16$

Fig. 4. BID for TDMA and IABA with $N_c = 4, 8$ and $r = 2, 8, 16$

$1) \cdot r$ and $BID^{TDMA}_{worst} = (N_c - 1) \cdot r + (r - 1)$, so $BID^{IABA}_{worst} \leq BID^{TDMA}_{worst}$, which means that TDMA is never better than IABA.

If for a given application, the exact bus cycle for each access to memory is known, we can use BID^{TDMA}_{exact} instead of the worst-case value. However, BID^{TDMA}_{exact} values can *only* be determined after *all* hardware is designed, when the application binaries can be generated and analyzed on the actual platform (including hardware and system software). Further, any source of uncertainty in the analysis of any component has an impact on BID^{TDMA}_{exact} accuracy.

Chip vendors decide which arbitration policy to implement, typically without knowing the applications that will run on top. In order to help a chip vendor to choose between TDMA and IABA, we can use an expected BID instead of the exact value for TDMA, by assuming that bus requests distribute uniformly over the TDMA window. Intuitively this should be the case, as one does not expect any particular distribution in time at which requests are ready with respect to the window. We verified this assumption on our processor setup, explained in Section III over all the EEMBC benchmarks. The difference between the maximum number of accesses done in any cycle of the window and the minimum number of cycles done in any other cycle (i.e. $\frac{min-Max}{min}$) is only: 3.5%, 2.4%, 1.8%, 0.7% and 0.6% respectively for windows of size 128, 64, 32, 16 and 8 cycles. Figures 3 show the distribution of accesses per cycle under different TDMA window sizes.

To calculate the expected value of the BID with TDMA, we note that, going back to Figure 2, TDMA windows of $N_c \times s$ cycles are comprised of two intervals in terms of BID: The first one of $s-r+1$ cycles with BID value 0, which corresponds to the cycles in which the request is in its slot. The second interval, containing the rest of the cycles, forms a linear decay that goes from the maximum BID value, i.e., $(N_c - 1)s + (r - 1)$ down to 1. The average expected BID for both intervals can be expressed as shown in Equation 3. In the first addend BID equals 0 for $s - r + 1$ cycles and the second addend corresponds to the linear decay, both addends divided by the number of cycles of the window, $N_c \cdot s$:

$$BID^{TDMA}_{expected} = \frac{0 \times (s - r + 1) + \sum_{i=1}^{(N_c-1)\cdot s+(r-1)} i}{N_c \cdot s} \quad (3)$$

Overall, Equation 3 allows us to compare IABA and TDMA for any setup including the core count accessing the bus and the bus latency. As an illustrative example, Figures 4(a)-(d) show different values for TDMA and IABA. Interestingly, in all cases the cross-point between $BID^{TDMA}_{expected}$ and BID^{IABA}_{worst} occurs when the slot size s is equal to $2 \cdot r$, i.e., TDMA offers better worst-case performance than IABA when the TDMA slot size $s < 2 \cdot r$, as we show in our experiments later.

III. Empirical comparison of TDMA and IABA

In this section we qualitatively and quantitatively compare TDMA and IABA using different metrics: required changes at hardware and software level (i.e. analysis tool), WCET estimates obtained, prioritization capabilities, average performance and time composability.

A. Experimental Setup

The quantitative metrics have been obtained using SoCLib [4] simulator. Based on a pipelined processor core comprising fetch, decode, execute and write-back stages, we model four and eight cores architectures in which each core has private instruction and data caches, which is common in current high-performance and real-time embedded processor designs [3] [12].

The cache hierarchy comprises first-level 4KB instruction cache and 4KB data cache. The shared L2 cache is 256KB in size with 8 banks and 8-way associativity. The cache line size is 16 bytes in all caches. Hits on the data and instruction caches take 1 cycle and misses 2 cycles. Hits to L2 take 3 cycles and misses 1 extra cycle. The access to main memory is 100 cycles. The bus request latency is 2 cycles.

With the aim of evaluating TDMA and IABA, we make the bus the only resource for which tasks in different cores compete. To that end, we deploy existing solutions that partition the L2 cache and the memory bandwidth. The former can be implemented using the bankization or columnization (way partitioning) techniques proposed in [16]. These techniques simply change some bits of the address of the requests sent by different tasks in each core such that the requests of each task access a different L2 cache bank in the case of bankization, and a different way in the case of columnization. The

memory bandwidth is partitioned using the techniques proposed for the memory controller in [17][6].

We used twelve benchmarks of the EEMBC Autobench suite [18] as reference for the analysis of single-path programs. This suite is well-known and reflects the current real-world demand of some embedded systems. We used: *a2time*, *aifirf*, *basefp*, *cacheb*, *canrdr*, *idctrn*, *matrix*, *pntrch*, *puwmod*, *rspeed*, *tblook* and *ttsprk*.

B. WCET analysability: feasibility and required changes at hardware level and in the WCET analysis tool

Next we show the main changes required by TDMA and IABA when using Static and Measurement-based timing analysis.

a) Static Timing Analysis (STA): STA techniques rely on (1) the construction of a cycle-accurate model of the system and (2) determining a trustworthy upper-bound on the WCET. At the hardware level, it is necessary for every resource to have a bounded access latency that is provided as input to the model. The same principle applies to the bus: On the one hand, the time it takes a request to access the bus must be boundable, even in the presence of inter-task interferences. On the other hand, the tighter this bound is, the tighter the obtained WCET estimates. If the bus uses IABA, no change is required in the STA tool: Basically, the access latency of each request to the bus is augmented with the BID^{IABA}_{worst}. IABA requires changes only in the configuration of the processor parameters given to the tool but not in the tool itself.

For TDMA, using BID^{TDMA}_{worst} as bound is pessimistic since $BID^{TDMA}_{worst} \geq BID^{TDMA}_{exact}$ for any given cycle. If the exact cycle in which each request accesses the bus could be known by the STA tool, we could use for each request its BID^{TDMA}_{exact}, thus eliminating the pessimism on the estimate. However, imposing that the STA tool provides the exact cycle in which each bus request occurs is complex [13] or even unfeasible [22]. In reality, for each request, rather than an exact cycle, it can be estimated a time interval [13] in which it can access the bus. This inaccuracy comes from the fact that the structure of the program includes branches, loops, etc., which complicate the estimation of exact cycles, since the time at which one request accesses the bus may depend on the path followed to get to the basic block in which the request is. Note that as soon as the estimated time interval for a request to access the bus is longer than the window, the STA tool has to use BID^{TDMA}_{worst} as the bound.

b) Measurement-Based Timing Analysis (MBTA): MBTA techniques rely on extensive testing performed on the real system, recording the so-called high watermark execution time, i.e. the longest observed execution time. The high watermark observed during this testing phase is multiplied by an engineering margin. The outcome is used as WCET estimate for the task during deployment time.

Both TDMA and IABA require hardware changes in order to enable the use of MBTA tools: With IABA, during the testing phase the program under study is run in isolation. Every time a request to the bus is ready it is artificially delayed by the architecture by BID^{IABA}_{worst} cycles, which requires changes in the bus arbiter [16]. From traces obtained during this execution in isolation, a WCET estimate for the task is obtained. At deployment time, the hardware is instructed not to introduce any artificial delay. The key point of this solution is that the artificial delay introduced during testing upper bounds the inter-task interferences that the task can suffer during deployment time [16].

For TDMA, it is critical that every request gets access to the bus exactly in the same cycle during both testing and deployment time, otherwise losing trustworthiness on the results. One possible solution is to synchronize the start cycle of a task with a fixed point of the TDMA window in each execution, by means of some hardware support.

C. Worst Case Performance

In this paper we use RapiTime [19], a commercial measurement-based WCET analysis tool. RapiTime uses path analysis techniques to build up a precise model of the overall code structure and determine which combinations of subpaths form complete and feasible paths through the code. RapiTime combines the measurement and control flow analysis information to compute measurement based WCET estimates in a way that captures the execution time variation on individual paths.

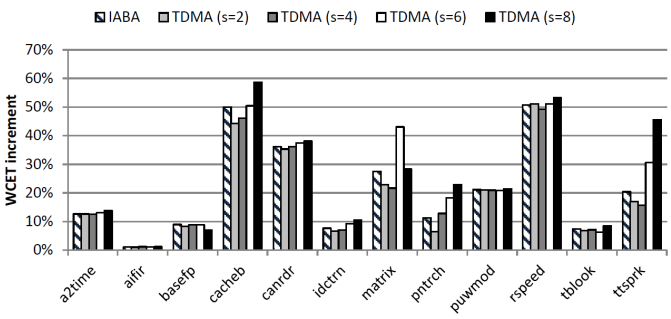
For TDMA, we assume two different scenarios: a scenario (1) in which the exact cycle of every bus access is known by the STA tool and the scenario (2) in which the exact cycle is unknown and we always have to assume the worst case to give safe bounds. Note that, these two scenarios do not affect the IABA case, which always assumes the worst case, but make a big difference in the case of TDMA.

Case (1) Exact bus access cycle known: Figure 5a shows the WCET degradation (increment) obtained for each EEMBC under a 4-core configuration using IABA and different TDMA setups. As a reference case, we use the WCET estimate obtained when the benchmark runs in isolation in the processor so $BID = 0$. We observe that some benchmarks are barely affected by the bus policy, like *a2time*, *aifir*, *basefp*, *idctrn*, *puwmod* or *tblook* due to their low memory traffic (most of them fit in the L1 caches).

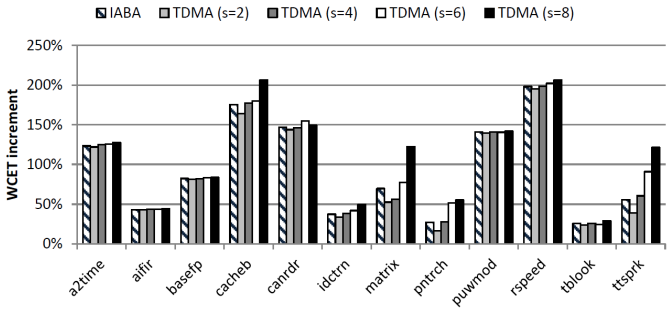
For the rest of the benchmarks, which show some WCET variability depending on the slot size, selecting the best TDMA slot size is tricky. A short slot size makes the overall window size smaller, so regardless of the cycle in which a requests is ready, it has to wait less to get access to the bus. On the other hand, long slot sizes favor bursty requests, i.e., several requests one after the other, since they can be processed consecutively.

In general, TDMA performs better with smaller slot sizes because the access latencies are smaller. More specifically, when slot sizes are such that $s < 2r$ (as we have seen in section II-C), which corresponds to $s = 2$ and 4 in our experiment, the WCET improvement is around 5% for memory-intensive benchmarks.

An interesting case is the *matrix* benchmark for which with $s = 6$ its WCET is 15% worse than when $s = 8$, $s = 2$ and $s = 4$. This high increment can be explained by the bursty access pattern generated by the benchmark: if more than one request fits into the slot, when we have a burst of requests the second and subsequent requests can access the bus immediately after the first one finishes. As a result, bursty requests take advantage of larger slot size. However, if the second and subsequent requests do not fit into the remaining cycles of the slot, they have to wait till the end of current slot and for other core slots, 3 in this case. The burst pattern generated by *matrix* fits when the slot size is $s = 8$, but not for $s = 6$. This behavior is magnified by the repetitive nature of the benchmark, exacerbating the WCET degradation.

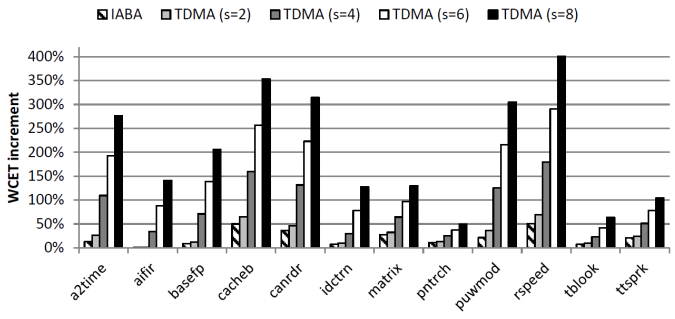


(a) $N_c = 4$ cores

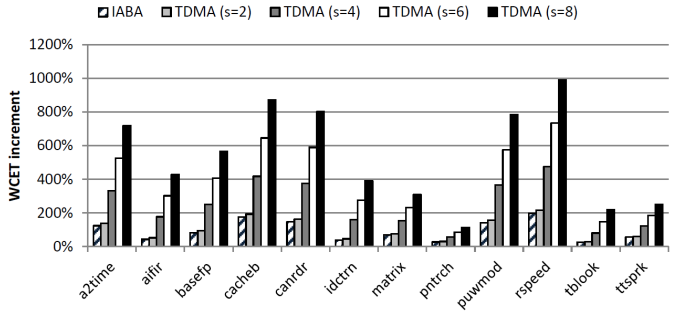


(b) $N_c = 8$ cores

Fig. 5. WCET increment when the bus access cycles are known for every request



(a) $N_c = 4$ cores



(b) $N_c = 8$ cores

Fig. 6. WCET increment when the exact bus access cycle is not known.

We also observe that IABA is close to the best TDMA setup despite the access time of all requests is known, which is useful for TDMA but not for IABA. On average TDMA leads to WCET estimates around 2% lower than IABA, being *cacheb* the benchmark leading to the highest difference (5.7%).

For the 8-core setup, shown in Figure 5b, we have observed the same trend. The main difference with respect to the 4-core setup is that the WCET variations are greater because of the higher number of bus contenders. We also observed that there is no benefit of that burst-like behavior mentioned before because the beneficial effect of augmenting the slot size is masked by the higher latencies due to the longer window size (now for the same slot size, the window size is doubled in comparison with 4-cores). We also observe the same trend as for 4-cores that for slot sizes $s < 2r$ TDMA performs better than IABA. More precisely up to 6% and 17% on average.

Case (2) Exact bus access cycle unknown: If the cycle in which requests arrive to the bus cannot be determined, for instance because the STA tool cannot be changed or because it cannot be determined with enough accuracy to provide any benefit, we have to use $BID^{worst}_{TDMA} = (N_c - 1) \cdot s + (r - 1)$ in the case of TDMA for STA. In general, the smaller the value of s is, the better the result. In the best case, $s = r$ and BID^{worst}_{TDMA} is $(N_c - 1) \cdot r + (r - 1)$. For our 4-core setup $BID^{worst}_{TDMA} = 3r + (r - 1) = 4r - 1$. With IABA, every request waits for all the other contenders to access the bus, i.e., $3r$. Figure 6a shows the WCET increment for the 4-core scenario. In general, we observe that IABA is better than TDMA in all cases (7.6% on average), being the improvement in the case of *puwmod*, *cacheb*, *canrdr*, *rspeed* and *a2time* higher than 10%.

For the 8-core setup, the same trend is observed. Figure 6b shows that IABA outperforms TDMA by up to 18% and 10% on average. Also for TDMA, as expected, the smaller the slot size the better since a larger slot size means also increasing the worst-case BID.

To complete our study, Figure 7 shows the WCET increment for the *ttsprk* benchmark as the percentage of known bus access times varies. This will be the case if the WCET analysis tool can not provide the exact bus access cycle for all the request but only for a percentage of the total number. The 0% case corresponds to the always worst-case scenario (Figure 6) and the 100% one to the exact cycle known scenario (Figure 5). We observe that the WCET increment for TDMA is inversely proportional to the percentage of access times known. We observe that at least 50% of the exact access times are needed to obtain some gains with TDMA. We observed similar trends when analysing the other EEBMC benchmarks.

D. Time-bounded Prioritization

From Figure 5, we observe that some tasks are not affected by the duration of BID while others are significantly affected. This knowledge can be used to reduce the WCET estimate for those tasks more affected by BID (e.g. *cacheb*) by granting them access to the bus more often, or prioritizing them [2]. We notice that in this context, prioritization does not mean to flush or stop in-flight requests, or to obtain the bus access immediately, it means only to have more resources assigned than other tasks.

With TDMA, prioritization is done by assigning different slot sizes to each task: long slots are assigned to the bus-intensive tasks, leaving the rest of tasks with shorter slots. For example, we can divide the window in one big slot and several small slots and assign the big slot to a memory-intensive task and the smaller slots to low memory demanding tasks.

With IABA, the resource management can be obtained by using a hierarchical round-robin policy, first proposed in [16] and analysed in [7]: Tasks are divided into groups and subgroups. For example, if

[2]This can be done at hardware level identifying each core, which means that we are not actually prioritizing tasks, but rather cores.

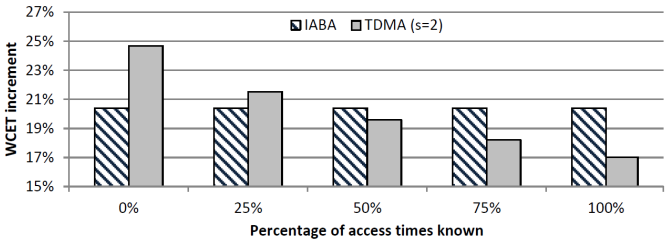


Fig. 7. WCET increment for *ttsprk* with different percentage of information.

we want to prioritize one task T_1 over other three tasks T_2, T_3 and T_4, we make 2 groups, the first one containing T_1, and the second group containing T_2, T_3 and T_4. Then the arbitration algorithm chooses one of the groups in a round robin fashion. Inside that group tasks are selected also using round robin. The result is that, applying the IABA philosophy, the high priority task, T_1 only has to wait for one round to get access to the bus (i.e. it obtains the bus once every two rounds), and the other three tasks, have to wait for the accesses of the other two tasks in the second group which are interleaved between the corresponding three accesses of the high priority task. Hence, T_2, T_3 and T_4 get access to the bus once every six rounds.

Overall, both IABA and TDMA enable thread prioritization. However, TDMA offers finer granularity than IABA. With IABA tasks can access the bus with a frequency limited by the number of groups and subgroups, and the number of tasks in each group and subgroup. For TDMA we can simply adjust the slot size for each of the tasks, at bus cycle granularity.

E. Time Composability

Time composability is a property that is becoming of paramount importance in the design and deployment of Integrated real-time systems such as IMA in avionics [25] and AUTOSAR [10] in the automotive domain. Time composability exists when the timing behavior of a certain component of a system can be derived in isolation and that timing behavior is not affected by the other components of the system. Time composability is a desirable feature of a system, because components of the system can be added, removed or upgraded without affecting the timing behavior of the other components of the system. This removes the need for analyzing, verifying and testing the whole system again when some components are changed, thus enabling incremental qualification and system upgrades. In this paper, we consider time composability in the sense that the WCET estimate for a task τ_i can be computed in isolation and it is not affected by the other tasks that may run concurrently with τ_i.

Note that time composability and prioritization are somehow opposed metrics. If we prioritize one task's accesses to the bus, we make this task faster, but the WCET estimates of the other tasks depend on the resources allocated to the prioritized task. It is up to the system designer to define a system cost function to determine which metric is more important, prioritization or time composability.

With TDMA, if we keep the window size fixed, time composability is not affected. For example, let's assume that the case of two tasks, T_1 and T_2, both having a slot size (s) of 5. Further assume that the bus latency of 1 cycle (r). Under this scenario, the worst-case BID is $(10-5)=5$ cycles, no matter what is the schedule of the rest of the tasks. However, if the window size depends on the other tasks' slot sizes, for example, T_2 increases its slot size to 6, which makes a window of $(6+5)=11$, then the worst-case BID for T_1 is $(11-5)=6$ which affects its WCET.

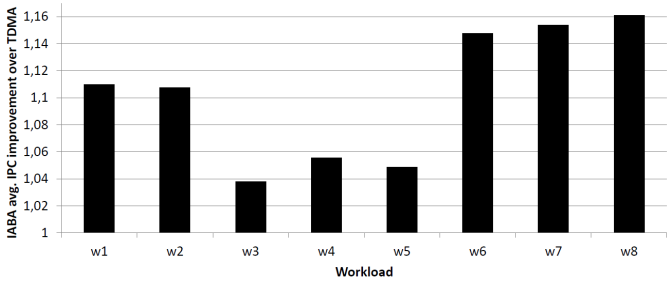


Fig. 8. Average performance improvement of IABA over TDMA

In the case of IABA, if we keep the groups and number of tasks within the groups fixed, time composability is not affected. Otherwise, for example, if we change one task with one priority for another one with different priority, then the WCET estimates for the other tasks change, thus losing the time composability.

F. Average Performance

In order to measure the average performance of IABA and TDMA, we run eight 4-thread randomly generated workloads under IABA and TDMA, comparing the performance of each benchmark in terms of IPC (*Instructions Per Cycle*) under both policies. Note that with TDMA, both at analysis and deployment time tasks are not allowed to use the bus out of their slots. With IABA, instead, at analysis time, each bus access of a task is artificially delayed by BID^{worst}_{IABA} cycles. At deployment time, when several tasks run simultaneously, no artificial delay is introduced. The only delay tasks suffer accessing the bus are due to actual inter-task interferences.

Figure 8 shows the average performance improvement of IABA over TDMA in terms of the average IPC improvement of all benchmarks in a workload. IABA provides better performance in all workloads than TDMA because it is a work-conserving policy. The improvement ranges from 4% to 16% with an average of 10%.

G. Summary and Discussion

Table II summarizes the different properties analyzed in this paper for TDMA and IABA. In this table, we use the following symbols: $++$ (very high), $+$ (high), $-$ (low), and $--$ (very low).

WCET tightness: Whether the exact cycle of every bus request is known or not, plays a key role in TDMA tightness, while IABA is insensitive to it. Although knowing the cycle in which requests are generated has been identified as difficult – or even impossible – to be obtained by a STA tool, we have used it as the best possible scenario for TDMA. Under this scenario, TDMA slightly outperforms IABA for small slot sizes, $s < 2r$, obtaining about 5% lower WCET estimates for 4 cores, for the most memory-intensive benchmarks. We have also seen that, in order to obtain some gains with TDMA, at least 50% of the bus request arrival times must be known. In the more feasible case that exact bus arrival cycles are unknown, IABA largely outperforms TDMA.

Average performance: Since IABA is a work-conserving policy, it achieves significantly higher performance than TDMA.

Time composability: Both techniques are equally time composable if no prioritization is performed across tasks.

Prioritization: In terms of resource management, TDMA offers better (finer) management granularity than IABA, because TDMA

TABLE II. SUMMARY OF THE PROPERTIES OF TDMA AND IABA

	WCET tightness	Avg perf.	Time composability	Prioritization	HW/SW changes
IABA	++	++	++	+	+
TDMA	+	−−	++	++	-

allows adjusting the slot of each contender at cycle granularity to match the application's requirements. Still IABA can use *grouping*, which imposes a coarser granularity for prioritization than that of TDMA.

HW/SW changes: While both techniques require hardware support, IABA does not require changes on the static timing analysis tools. If those tools have to provide bus access time intervals for TDMA, they must be deeply changed.

IV. BENEFITS FOR INDUSTRY

Since several processors use buses as the main communication channel, the study carried out in this paper provides valuable information for real-time system designers. In particular, this paper helps hardware and software designers in early design steps to choose which policy best fits their needs.

For hardware designers, at the time the chip is being designed, the information about the target applications may not be accessible. It can also be the case that the chip targets different real-time markets where applications have different profiles in the use of the shared resources including the bus. Our analysis shows that IABA is the best choice because it provides better results and puts much fewer requirements on the information needed from the software designer to provide tight WCET estimates.

If the application environment is known and if software designers can influence the chip design, software designers can decide whether TDMA or IABA is better. In particular our study will help them determining whether the WCET estimate reduction obtained with TDMA (when the exact cycle in which each bus access occurs) pays off the effort of determining those exact cycles or if, instead, IABA is accurate enough. The BID (obtained with our analysis) of the arbitration policy chosen can be provided to the early-stage timing analysis tools, usually called Code-Level Timing Analysis (CLTA) tools, to determine the WCET tightness with TDMA and IABA.

V. CONCLUSIONS

Bounding the effect of inter-task interferences is of paramount importance to provide meaningful WCET estimates in multicore processors for CRTES. In this paper, we have evaluated and compared the two most used bus arbitration policies, TDMA and IABA, intended to deal with inter-task interferences in On-Chip buses. Concretely, we have seen that both policies can provide WCET analyzability and time composability, since WCET estimates for applications can be computed in isolation regardless of the workload executed concurrently. IABA presents worse prioritization capabilities and better WCET and average execution time with little burden for the user.

VI. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Space Agency under NPI Contract 40001102880, the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA) and the Ministry of Science and Technology of Spain under contract TIN-2007-60625. Eduardo Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI2009-05455.

REFERENCES

[1] *AMBA Bus Specification.*

[2] *ESA contract: 22279/09/NL/JK*.

[3] *NGMP Preliminary Datasheet Version 1.6, August 2011*.

[4] SoCLib. http://www.soclib.fr/trac/dev.

[5] *ARC Advisory Group. Process Safety System Worldwide Outlook. Market Analysis and Forecast through*. 2012.

[6] B. Akesson, K. Goossens, and M. Ringhofer. Predator: a predictable SDRAM memory controller. In *CODES+ISSS*, USA, 2007. ACM.

[7] R. Bourgade, C. Rochange, M. De Michiel, and P. Sainrat. Mbba: A multi-bandwidth bus arbiter for hard real-time. In *Embedded and Multimedia Computing (EMC), 2010*, pages 1–7, 2010.

[8] P. Clarke. *Automotive chip content growing fast, says Gartner (9/6/2010)*.

[9] D. Dasari and V. Nelis. An analysis of the impact of bus contention on the wcet in multicores. In *HPCC-ICESS*, pages 1450–1457, 2012.

[10] M. Di Natale and A. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 2010.

[11] G. Heiser. The role of virtualization in embedded systems. In *1st Workshop on Isolation and Integration in Embedded Systems*, pages 11–16, Glasgow, UK, Apr 2008. ACM SIGOPS.

[12] Infineon. AURIX Safety joins Performance.

[13] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. Bus-aware multicore WCET analysis through TDMA offset bounds. *Real-Time Systems, Euromicro Conference on*, 2011.

[14] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[15] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *RTSS*, 1999.

[16] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, Austin, TX, USA, 2009.

[17] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero. *An Analyzable Memory Controller for Hard Real-Time CMPs* . ESL, 2009.

[18] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[19] RapiTime. *www.rapitasystems.com*, 2008.

[20] 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*, 2007.

[21] E. Salminen, T. Kangas, V. Lahtinen, J. Riihimäki, K. Kuusilinna, and T. D. Hämäläinen. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *J. Syst. Archit.*, 53(8), Aug. 2007.

[22] J. Staschulat and M. Bekooij. Dataflow models for shared memory access latency analysis. EMSOFT '09.

[23] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian. Towards scalable, energy-efficient, bus-based on-chip networks. In *HPCA*, 2010.

[24] E. Wandeler and L. Thiele. Optimal TDMA time slot and cycle length allocation for hard real-time systems. In *ASP-DAC*, 2006.

[25] C. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. *DASC*, 2007.

[26] M.-K. Yoon, J.-E. Kim, and L. Sha. Optimizing tunable WCET with shared resource allocation and arbitration in hard real-time multicore systems. In *RTSS 2011*.