Adaptive TDMA bus Allocation and Elastic Scheduling: a unified approach for enhancing robustness in multi-core RT systems

Paolo Burgio[†], Martino Ruggiero[†], Francesco Esposito[§], Mauro Marinoni[§], Giorgio Buttazzo[§] and Luca Benini[†]
[†] DEIS - University of Bologna - Italy
[§] ReTiS - Scuola Sup. S. Anna Pisa - Italy
{*paolo.burgio, martino.ruggiero, luca.benini*}*@unibo.it*
{*f.esposito, m.marinoni, buttazzo*}*@sssup.it*

***Abstract*— Next-generation real-time systems will be increasingly based on heterogeneous MPSoC design paradigms, where predictability and performance will be key issues to deal with. Such issues can be tackled both at the hardware level, by embedding technologies such as TDMA busses, and at the OS level, where suitable scheduling techniques can improve performance and reduce energy consumption. Among these, elastic scheduling has been proved to provide satisfactory results by dynamically reducing task periods at run-time to ensure the highest utilization possible of the processors. On the other hand, elastic scheduling lowers the degree of predictability and increases the complexity of the analysis at the system level. This reduces the benefits given by the TDMA bus, which relies on the high level task analysis for a robust and efficient slot allocation. Starting from this consideration, we propose a system where the elastic scheduling and the TDMA bus work synergistically. We introduce a QoS-aware adaptive bus service which takes the best of both techniques, mitigating their drawbacks at the same time. We show how the overhead introduced by coordination action is small, and it is however dominated by the benefits of the overall strategy in terms of performance and predictability guarantees.**

I. Introduction

Key concerns in real-time applications are safety, quality and cost. System architectures targeting industrial sectors like automotive, medical and avionics need to provide strong guarantees in terms of predictable behavior and reliability. Multiprocessor systems-on-chip (MPSoC) will be increasingly used in these application domains to meet the tight cost and energy efficiency constraints. Future platforms will indeed embed several general purpose processors and few dedicated HW coprocessors for those critical functions requiring high performance levels [1].

Predictability and modularity of such MPSoCs are still an open issue in the research community [2]. The key challenge is mainly represented by the shared hardware resources, such as the system interconnect. The system bus is shared among multiple communication actors (cores, IOs, accelerators, etc.) thus introducing contention which leads to potentially unpredictable response times. The scenario of simple systems with only one bus master can be precisely analyzed. However, more masters contend for bus access, the more difficult it is to analyze the traffic on the bus and the more conservative will be the bounds on latency that can be guaranteed. This situation becomes even worse considering sporadic task activations.

TDMA-based bus arbitration is frequently used in current MPSoC architectures to obtain a predictable system [3], [4], [5], [20]. TDMA slot dimensioning is one of the most important parameters in this case, since the bus has to guarantee the satisfaction for the bandwidth required by every task running on the platform. However, real-time applications are becoming increasingly complex. Tasks running on CPUs generate heterogeneous traffic patterns with different level of priorities and real-time requirements, which make the analysis of the entire system very complex. The problem of TDMA slot dimensioning has been already tackled in the real-time community, but all of these solutions are based on a top-down approach which assumes a calibrated, accurate model of the overall system. Obtaining a good abstraction of both hardware and software is very difficult and costly, and it requires a high standard of modelling experience and validation.

The increased hardware complexity and the plurality of the external inputs that could modify the execution pattern effect the worst case execution time (WCET) analysis producing very pessimistic results. Sometimes it is preferred to relax some bounds and manage an infrequent overload condition with a specific support. The most well known techniques include: the possibility to split the job in a mandatory and an optional part [6], [7], the possibility to skip some jobs following a predefined rule [8], and the expansion of tasks periods [9], [10] (the so-called *Elastic Scheduling*). These techniques reduces the workload on a single core, but a multi-core scenario, they have also impact on the behaviour of the entire system due to inter-dependencies between the task and cores. These dependencies can both be explicit, due for instance to control flow and synchronization between tasks, and implicit, such as shared resource contention. This makes an off-line WCET analysis very complex and in RT systems may result in a loss of performance.

The main contribution of this paper is represented by a novel solution to the problem of TDMA slot dimensioning. Our target platform is a multiprocessor system-on-chip composed by general-purpose cores. The considered TDMA slot scheduling is a periodic wheel with one slot for each

master in the system. We propose a new system where the Elastic Scheduling and the TDMA bus work synergistically to ensure the highest utilization of the processors even in case of dynamic variations of the workloads at run-time. When a processor is subject to a workload change it makes a request to adapt its share of bus bandwidth. A layer is needed to mediate all the requests, and we adopted a centralized approach, where a master core is appointed to collect all requests and compute a fair redistribution of the bus. Based on the new bus allocation, each core uses the elastic algorithm to reach the desired utilization.

Different and complex algorithms have been proposed in related works, but none of them is at the same time dynamic and application QoS-aware.

This paper is organized as follows. Section II gives an overview on related works. In Section III the target architecture is described and elaborated. Section IV presents the proposed algorithm for TDMA slots assignment and task scheduling. In Section V, the proposed virtual platform environment is described. Section VI presents and analyzes the experimental results of our design space exploration. Finally, Section VII presents the conclusions.

II. Related Works

Although TDMA is a widely adopted technology in MPSoCs, dynamic reconfiguration and run-time re-allocation of the time slots for shared resources access (being them shared busses, memories or other) were not intensively studied. On the other hand, some interesting works on more general approaches (for instance, involving other arbitration schemas) can be found.

In [11] a ring-based infrastructure for a programmable fixed priority arbiter is shown. The priority at which requests are processed is driven via directly programmable control registers. This is a good and lightweight solution and a similar approach can be also found in [12], where authors introduce a very general register based architecture for TDMA communication infrastructure, as well as an algorithm for efficiently programming the hardware registers.

In [13], a pure dynamic TDMA bus (called dTDMA) architecture is shown. Authors propose a reactive slot allocation occurring whenever a new request of the shared resource is issued. Their schema provides a fair allocation (i.e. each requestor is given the same bandwidth) and thus ensures a good predictability level of the entire communication system. On the other hand it is not suitable for our purposes since the arbiter is not aware of QoS needs. At the same time is at a really fine temporal grain, thus introducing an overhead we can not tolerate.

Authors in [14] used an additive bus model which can be applied with relatively good approximations only if the bus load is kept below a certain threshold. Even in the case of such low bus utilization, no strong guarantees regarding QoS can be provided. Authors in [15], [16], [17] presented several bus access optimizations for enhancing predictability in MPSoCs, but none of them has been demonstrated and validated on a real platform target, hence their modeling abstractions have not been fully validated.

Support for variable workload and management of overload conditions has been studied in the real-time community for a long time. Different approaches have been proposed to suit different application fields and to meet particular constraints. Among the others, the mostly adopted are:

- *Imprecise computation*: Each job is divided in a mandatory part ad an optional one. The optional part can be skipped if the core workload exceeds a certain threshold. [6] [7]
- *Job skipping*: It is possible to reduce the workload by skipping some job instance in each task. Hamdaoui et al. [8] propose an algorithm that does this in a fair way and at the same time guarantees a minimum number *m* of job instances in a windows of *k* periods.
- *Elastic task scheduling*: The last technique is based on the modification of task periods. Authors in [18] proposed an algorithm that manages periods in the taskset like a set of springs.

However, all these approaches are based on a fixed platform giving only one possible WCET for each task and work in order to reduce the workload on a single core. Investigation on the possibility to spread the overload among cores redistributing the the access on the shared resources, like the bus, has not been done yet.

III. Target Architecture

Future real-time architecture systems will be Multiprocessor System-on-Chip (MPSoCs) composed, among the others, by the following building blocks:

1) Several processing units with very simple micro-architecture (i.e. with no branch prediction or multi-threading), with both instruction and data caches;
2) A highly predictable interconnection, such as a TDMA-based shared bus or NoC, which at the same time can provide the performance required by applications.

As regards as the software layer, each core has its dedicated application and real time OS image in its private memory. This architecture depicts the scenario of highly

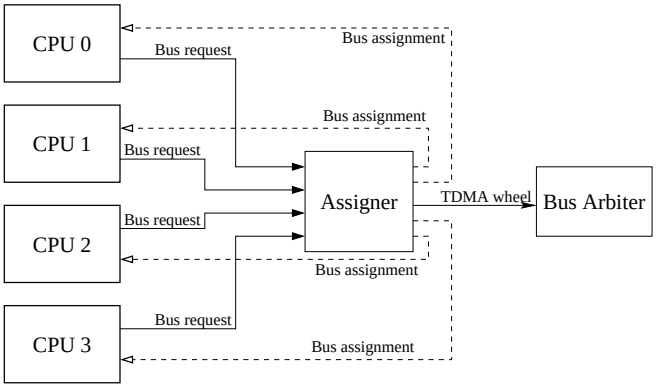

Fig. 1. Algorithm communication structure

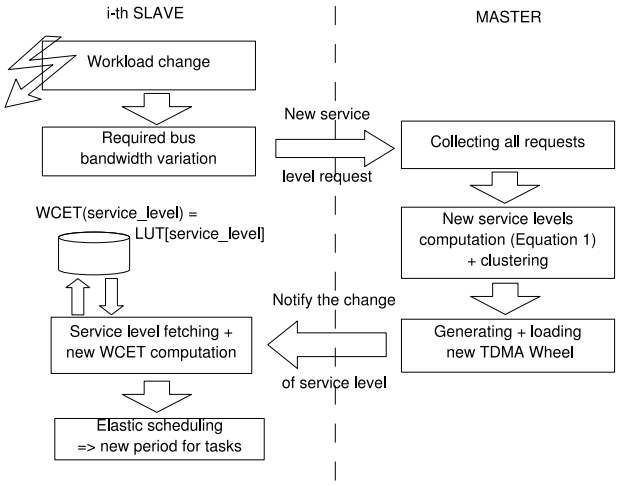

Fig. 2. Bus bandwidth assignment algorithm

parallelized applications, where large amounts of data are elaborated distributing the workload on multiple symmetric cores. Moreover, we assume the presence of sporadic tasks activated by external interrupts due to the interaction with sensors or users. This makes the overall system increasingly complex as the number of cores grows. The static analysis is no longer efficient and the overall elastic scheduling approach deteriorates its performance, lowering the benefits.

IV. Bus bandwidth assignment algorithm

The algorithm presented in the paper works as a bridge between hardware and software in order to allow an assignment of the bus which is aware of the core QoS requirements. The communication structure is presented in Figure 1. Due to its boundary position, the Assigner could be implemented both in hardware or in software. We consider the latter situation.

In Figure 2 we show the algorithm in details. During system execution, a core may face a need for extra bus bandwidth, due for example to workload changes or to activation of sporadic tasks. Consequently the core asks the Master Core of the system for a certain (typically higher than the current) service level (R_i) for communication. The algorithm supports a discrete number of service levels (they are shown in Table I), each service level corresponds a certain bus bandwidth percentage. Clearly, the relation between service levels and bus bandwidth depends on to the number of cores in the platform and it is calculated off-line. At predefined instants, the Master fetches all new bus bandwidth requests coming from the other cores, mediates between them and recomputes the percentage (S_i) of the bus assigned to each core as

$$S_i = \frac{R_i}{\sum R_I} \quad (1)$$

and then generates a new Time Wheel. Clearly, the TDMA slots are set in order to assign the computed bus bandwidth to the cores. Our algorithm is not guaranteed to find the optimal solution but rather a fair tradeoff betweeen all requests, being at the same time extremely efficient and lightweights. The actual service level may be different (i.e. lower) than the one requested if multiple requests happen at the same size since the algorithm mediates between all of them. Moreover, a core performing no request may see its service level changing as an effect of a new scheduling due to other cores requests. Then, the new Time Wheel is loaded in the Bus Arbiter and the new service levels are notified to the cores.

0	ZERO	4	HIGH
1	MINIMUM	5	MAXIMUM
2	LOW	6	EXTREME
3	MIDDLE		

TABLE I
SERVICE LEVELS

Since this change implies a variation in tasks execution times, task periods have to be recomputed according to the Elastic Scheduling algorithm (described in Section V-A), using as input the WCETs of the task-set. It is important to remark that TDMA-wheel switching does not compromise the feasibility of a task-set running in a core characterized by a short TDMA slot assignment. In fact the actual task parameters (i.e. elastic constants, T_{min}, T_{max} and the deadline equal to the period) are defined off-line, in such a way to guarantee a feasible solution to the algorithm in all possible scenarios.

The WCETs depend on the bus bandwidth assigned to the core, so they also have to be recomputed. The complexity of WCET analysis techniques makes unfeasible to do this at run-time, so they are computed offline and stored inside a lookup table (LUT) to make them available to the cores. This storage area has to fill the smallest possible space. This is obtained providing only WCETs for the limited number of service levels and imposing a quantization to the values obtained with Equation 1.
The LUT size is a tradeoff between the memory space used and the obtained bandwidth granularity. Increasing the number of levels allows the algorithm to better fit cores requests and leads to a solution with an higher quality of service. On the other hand, each extra level means an increasing in the algorithm overhead, that is we need more space for storing information and a higher computational effort for execution times calculation. A fair tradeoff already happens with a small number of levels. With the hardware used in the experiments that will be presented, we empirically found 7 as an adequate value by a preliminar set of experiments.

After chosing the number of allowed bandwidth assignments (that is, the number of rows in the table), there are several options for dimensioning their values. The simplest is based on homogenousity: divide the valid bandwidth range by the number of elements. More sophisticated approaches minimize a defined metric: an example could be the aggregated bandwidth waste, i.e. is the sum of all quantization losses. In this work we adopted the homogeneous bandwidth division. Each row is composed by the WCETs of all tasks working with the selected bus bandwidth assignment. The WCET values can be obtained using a static code profiler

and analysis tool such as [19].

Once the WCETs have been loaded, tasks periods can be accordingly adjusted to meet real-time requirements and tasks can be now scheduled. The overall approach gives two main benefits: first the bus TDMA allocation is QoS-aware and secondly the OS scheduler can take more accurate decisions based on the bounds given by the dynamic TDMA arbitration policy.

V. Robust reconfigurable RT platform

In [20] a functional model of the target architecture described in Section III was developed for enabling in-depth architectural exploration. All cores are 32-bit ARM-based with an associated L1 cache. They are connected to a shared L2 memory via the shared bus. The L2 memory is segmented, i.e. there is a private portion associated to each core and a shared portion they can use for communication or data passing. An interrupt device is provided as well as a semaphore memory, a special memory device capable of test-and-set read operations. The latter is used for synchronizing concurrent accesses to shared resources, while the former provides the capability of efficiently propagating notifications/events in the system. The full architecture is shown in Figure 3. The communication bus is modelled

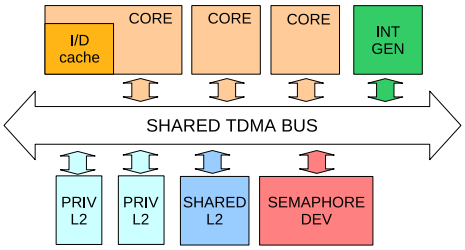

Fig. 3. Reference architecture

at transactional level (TLM) and takes into account features of modern high-performance communication buses (such as AMBA AXI [21] or ST StBus Protocol [22], [23]), namely the capability of supporting burst interleaving, multiple outstanding transactions and split transfers. The bus model is packet-based, i.e. a "transaction" on the interconnect is composed by several packets. A functional TDMA arbiter is implemented. It loads the so-called Time Wheel (in literature it is also referred to as *Slot Table*) from a text file. The Time Wheel contains all the information on a single TDMA Round and unrolls over the time line, repeating infinitely during the entire simulation.

As regards as the software running on the simulated hardware platform, it is possible to run stand alone (i.e. without the support of an OS) or ERIKA OS-based applications. ERIKA is an open-source (GPL2) multi-processor real-time operating system (RTOS) kernel, implementing a collection of Application Programming Interfaces similar to those of OSEK/VDX standard [24] for automotive embedded controllers. ERIKA is available for several hardware platforms and introduces innovative concepts, real-time mechanisms and programming features to support and exploit the micro-controllers and multicore systems-on-a-chip. With multiprocessor hiding, it is possible to seamlessly migrate application code from a single processor to multiprocessors without changing the source code. Retargeting an application from single to multiprocessor architectures only requires minor modifications at the configuration files, but allows retaining the source code. The main ERIKA features related to this work are: task scheduling according to fixed and dynamic priorities; interrupt handling for urgent peripherals operation (interrupts always preempt task execution); resource sharing with Immediate Priority Ceiling protocol.

A. Software support to dynamic bus assignment

To cope with overload conditions we extended the ERIKA scheduling support adding an implementation of the Elastic scheduling algorithm [18] where each task is considered as flexible as a spring, whose utilization can be modified by changing its period within a specified range. More specifically, each task is characterized by four parameters: a worst-case computation time C_i, a minimum period $T_{i_{min}}$ (considered as a nominal period), a maximum period $T_{i_{max}}$, and an elastic coefficient E_i. The elastic coefficient specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration: the greater E_i, the more elastic the task. In the following, T_i denotes the actual period of task τ_i, which is constrained to be in the range $[T_{i_{min}}, T_{i_{max}}]$. Moreover, $U_{i_{max}} = C_i/T_{i_{min}}$ and $U_{i_{min}} = C_i/T_{i_{max}}$ denote the maximum and minimum utilization of τ_i, whereas $U_{max} = \sum_{i=1}^{n} U_{i_{max}}$ and $U_{min} = \sum_{i=1}^{n} U_{i_{min}}$ denote the maximum and minimum utilization of the task set. The algorithm works on top of different scheduling algorithms with both static and dynamic priorities. For simplicity, in this paper tasks are scheduled by the Earliest Deadline First algorithm [25]. Hence, if $U_{max} <= U_d <= 1$, all tasks can be created at the minimum period $T_{i_{min}}$, otherwise the elastic algorithm is used to adapt the tasks periods to T_i such that $\sum \frac{C_i}{T_i} = Ud \leq 1$, where Ud is some desired utilization factor. It can easily be shown (see [18] for details) that a solution can always be found if $U_{min} \leq U_d$. If Γ_f is the set of tasks that reached their maximum period (i.e., minimum utilization) and Γ_v is the set of tasks whose utilization can still be compressed, then to achieve a desired utilization $U_d < U_{max}$ each task has to be compressed up to the following utilization:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_{max}} - (U_{v_{max}} - U_d + U_f)\frac{E_i}{E_v} \tag{2}$$

where

$$U_{v_{max}} = \sum_{\tau_i \in \Gamma_v} U_{i_{max}} \tag{3}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}} \tag{4}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i. \tag{5}$$

If there exist tasks for which $U_i < U_{i_{min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{max}}$ (so

that $U_i = U_{i_{min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and equation (2) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if the desired utilization U_d is greater than or equal to the minimum possible utilization $U_{min} = \sum_{i=1}^{n} \frac{C_i}{T_{i_{max}}}$, the iterative process ends when each value computed by equation (2) is greater than or equal to its corresponding minimum $U_{i_{min}}$.

B. Programmable TDMA Arbiter

From the implementation point of view, the main task of this work is the development of a communication protocol to let the simulator infrastructure aware of application level quality-of-service requirements. To achieve this goal, we extended the existing TDMA bus arbiter to be directly programmable at application level. This feature is split in an hardware and a software part and, according to this approach, the implementation process itself was split in two parts:

1) The definition of a high-level protocol for TDMA Wheel switching and the implementation of the APIs/system calls for the communication between the application layer and the simulation infrastructure. These low-level software calls must not introduce a significant overhead.
2) The extension of the existing arbiter model by embedding the state registers and implementing the corresponding handling logics, TDMA Slots updating and the Time Wheel loading (updates being applied) mechanisms.

Figure 4 gives an overview of how this new feature works and how it was implemented As shown, it is possible

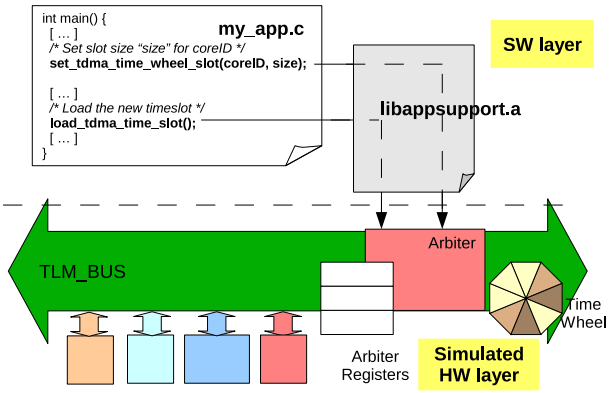

Fig. 4. Programmable TDMA Arbiter dual HW/SW structure

to set dynamically the size of a time slot (expressed in nanoseconds) via a call to the *set_tdma_wheel_slot*, which accepts also the ID of the master which is involved. When all the slots have been loaded, the entire table is marked as ”loadable” via a call to the *load_tdma_time_wheel* function and it will be loaded as the period of the old table expires. Of course a function is provided to read the value of the slot that was just written. If no value is set for the new slot of a certain master, we assume the previous value still holds, and it will be copied ”as is” in the new table. When a new table is loaded, an event is propagated in the system so to notify the change to the cores. This event is an interrupt call.

C. Communication support

Since the protocol implies high level coordination and communication between the cores, a support mailbox-like software feature was implemented. Two separate mailboxes were implemented (according to a request/grant protocol) and they reside in the shared portion of the L2 memory. When a processor needs more bandwidth it sends a request to the master processor a writing the desired service level in its Request Mailbox (*req_mailbox_write*). A soon as the master fetches (call to *req_mailbox_read*) and serves the request, it writes back in every core-associated Response Mailbox the assigned service level via a call to *res_mailbox_write* so that the core can fetch the actual value via a call to *res_mailbox_read*. The Time Wheel programming protocol and the communication infrastructure previously described must be as light as possible i.e. shall not introduce too much overhead. This is the reason why we write in the arbiter registers the Time Slot sizes expressed as nanoseconds instead -for instance- as bandwidth percentage. Since this is exactly the internal arbiter representation, it implies no other transformation/processing and can be instantly handled with no additional overhead. Again, the communication protocol consists on a single write operation, followed by a read after the interrupt notifies that the table was loaded. These are very lightweight operations and are slightly influenced by the TDMA scheduling even in case the working core was assigned a low bandwidth on the bus.

VI. Experimental Results

For the experimental setup we consider a task-set composed by avionics tasks, automotive tasks and memory intensive access tasks. For the avionics case we adopt the Matlab U.S. Navy’s F-14 Tomcat aircraft control task [26] that guarantee the aircraft to operate at a high angle of attack with minimal pilot workload; as automotive task we have chosen the coremark [27], a well known and widely used benchmark in the domain of embedded systems; finally a task that performs mathematic operations such as summation and characterized by intensive memory access. Each task-set is composed by a combination of these tasks, EDF is chosen as scheduling policy and no precedence constraints nor critical section has been considered between tasks.

Figure 5 show the behavior of a system composed by 2 cores: CPU0 is the master core and CPU1 is the slave one. Three tasks run on each core. The y-axis reports the computation time of each task while the x-axis reports the current time. Approximatively between 100 and 200 millisecond a request of additional TDMA slot bandwidth is requested by CPU1. This request is equal to the HIGH level among the service levels available; CPU0 does not request for additional bandwidth. Such a request lead to a rebalancing of TDMA bus slots by the master core. Starting from this moment the computation time of each task running on CPU1 improves while the corresponding one on CPU0 get

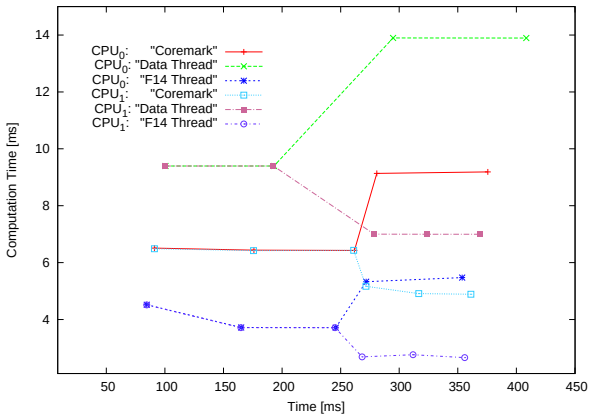

Fig. 5. Computation Time: CPU_0 vs CPU_1.

worse. The request, triggered by the third job of the avionics task running on CPU1, is made between 100 and 200 millisecond and the advantages for the CPU1 can be already appreciated in the fourth job for avionics and automotive tasks, and from the third job in case of the mathematical task.

Under these conditions, the task-set experiences a variable range of computation times: from 2 milliseconds for the avionic task up to 15 millisecond for the mathematical task. We performed measurements to catch the overhead introduced by our algorithm on the avionic task. The average overhead introduced is less than 5% of the computation time of the task itself. This overhead can be divided in a negligible (less than 5 microseconds) part we spent for the OS context-switch, while the majority of it is equally spent by the elastic manager to collect cores requests and accomplish task period variations, and to update the TDMA-wheel reallocation, i.e writing of the new values in arbiter registers and triggering the table switch. Moreover, the code to accomplish these tasks could be further optimized: for instance, the calculation of the new task periods has no FPU support which could instead provide a further improvement of the overall performance. However, even this not optimized version of the code has an execution time which is less than 100 times the basic context switch.

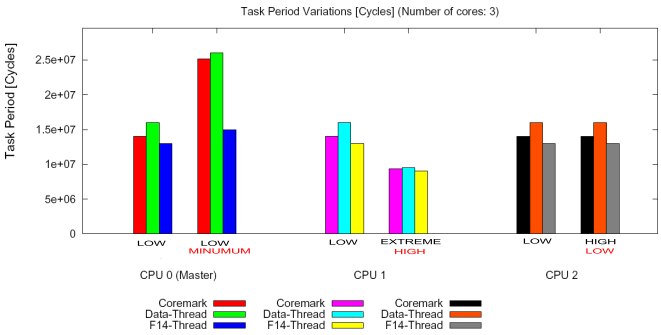

Fig. 6. Task Period Variations [Cycles] (Number of cores: 3).

A. Bus Access Time and Periods

In Figure 6 we show the variation of task periods. This is the case for three cores and three tasks for each core. As usual, CPU0 represents the master core while CPU1 and CPU2 are the remaining slaves that compose the systems. The amplitude of the histogram bars indicates the periods of the tasks and they are collected in three clusters, one for each CPU. Inside each cluster it is possible to appreciate the value of each task period corresponding with the old (on the left), requested (on the right) and actual (in red) service level. The system starts with a fairly distributed level of service equal to LOW, the corresponding TDMA slot assignment is 33% for each CPUs. This scenario is represented by the first set of bars inside each cluster. The second set of bars inside each cluster shows that CPU1 and CPU2 ask for additional bus bandwidth, respectively an EXTREME and a HIGH level of service, while CPU0 (master core) makes no requests. According with this set of requests, the master core assigns the Time Wheel the following way: MINIMUM (11%) for itself, HIGH (55%) for CPU1 that ask for the highest level of service and LOW (33%) for the CPU2. Note that with this particular combination of requests CPU2 is not able to improve its bandwidth and, despite of its request, it holds the initial percentage of TDMA bandwidth. This case has been deliberately chosen to highlight that requests for additional bandwidth must be considered as part of the whole set of demands coming from all CPUs. Figure 7

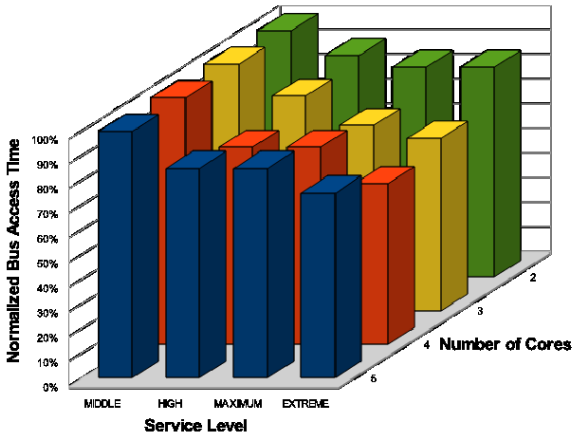

Fig. 7. Bus Access Time Percentage Improvements.

provides an exhaustive representation of the system response in terms of bus access time as a function of the number of CPUs and service level requested. For each measure, the values are normalized over the case with the same number of CPUs and the service level equal to MIDDLE. As usual we evaluate a system characterized by a composite task-set and we collect the response of a single measured CPU, that ask for different service level, in a multiprocessor context with variable number of CPUs. The service level of the analyzed CPU starts from MIDDLE up to EXTREME, while the whole number of cores that compose the system varies from two up to five. The system starts with a fair bus assignment

(MIDDLE service level) to the CPUs. The only CPU licensed to ask for different service level is the measured CPU, the remaining ones hold the initial service level (MIDDLE). The figure shows the improvements experienced by the measured CPU: the access time decrease if the number of CPUs or the service level requested increase. This trivial result is shown with the purpose of quantifying the advantage in terms of latency for the each bus access from the single CPU point of view, compared to the case of a static and fair assignment: same bandwidth for each CPU.

B. Quality of Control index

In control applications the performance of a periodic control task is a function of the activation period. Increasing the task activation period leads to a performance degradation, which is typically measured through a Performance Index $J(T)$ [10], [28]. Often, instead of using the performance index, many algorithms use the difference $\Delta J(T)$ between the index and the value of the performance index $J*$ of the optimal control. Many control systems belong to a class in which the function expressing the degradation is monotonically decreasing, convex and can be approximated as

$$\Delta J(T_i) = \alpha_i e^{-\frac{\beta_i}{T_i}}$$

where the magnitude α_i and the decay rate β_i characterize the single task. The evaluation of the whole task set is computed as

$$\Delta J = \sum_{i=1}^{n} w_i \Delta J(T_i) = \sum_{i=1}^{n} w_i \alpha_i e^{-\frac{\beta_i}{T_i}}$$

where the w_i are used to characterize the relative importance of the tasks.

To have a common scale for all task sets, the Quality of Control index used in this paper is expressed as

$$QoC = \frac{\Delta J_{nom}}{\Delta J}. \quad (6)$$

where ΔJ_{nom} is the value of the index calculated when tasks run at their nominal periods. A value of 1 means that all tasks are running with nominal periods.

All coefficients α_i and w_i are set to 1 for simplicity, while β_is are set to 20 in order to use the whole range [0,1] of the QoC index.

Taking the previous example (shown also in Figure 6), the values of ΔJ for CPU0 changes from 2.4 to 2.6 due to the Time Wheel variation. In Table II are presented the value of QoC for different approaches computed for the same example and normalized over the difference between$J_{t_{max}}$ and $J_{t_{min}}$. where $QoC_{t_{min}}$ is the best possible QoC (ΔJ_{nom}) obtained with a set of CPUs each with a dedicated bus. This case represents the virtual upper bound, but it is not really experienced, because we are working in a multiprocessor environment with shared communication bus; QoC_{dyn} is obtained adopting our run-time algorithm; QoC_{fair} is the result of a fair TDMA scheduling, where all the slots have the same size. This is also the starting point in our experiments, before the slot and task periods are modified. $QoC_{min_{bwd}}$ is a virtual QoC in case each core is assigned a low bandwidth (10% of the TDMA Round). This situation is typical of systems where each CPU must guarantee the timing constrains even with a small static slot assignment. Finally, $QoC_{t_{max}}$ is the QoC provided by the system if the longest allowed period is chosen for each task on every CPU. Table II shows that a system with the capability to dynamically adjust TDMA slots is able, starting by a fixed TDMA allocation, to have an improvement from 27% up to 31% of the QoC index. Moreover, the introduced overhead has negligible effect on the QoS (as prevoiusly said, in the average case it is around 5% of the computational time for the fastest task). On the contrary, a system characterized by a standard TDMA slot assignment is forced to operate with $QoC_{min_{bwd}}$, due to a lower bus-access-time.

$\Delta QoC_{t_{min}}$	1.0000
ΔQoC_{dyn}	0.7067
ΔQoC_{fair}	0.6947
$\Delta QoC_{min_{bwd}}$	0.6357
$\Delta QoC_{t_{max}}$	0.5925

TABLE II
QOC INDEXES.

VII. CONCLUSIONS

In this paper we presented an algorithm for the sizing of TDMA slots for concurrent bus accesses and task period dimensioning. The target architectures are RT MPSoCs where running tasks face unpredictable situations (external interrupts, interaction with users) and thus the standard off-line WCET analysis techniques are no longer efficient. This results in a loss of accuracy and consequently a loss of performance both of the TDMA bus scheduling and Elastic Scheduling, which cannot work at their best. We proposed a system where the shared bus arbiter works in coordination with the Elastic Scheduling algorithm of the OS, so both the TDMA Time Wheel and task periods are adjusted at run-time to meet the performance constraints. The algorithm is aware of task-level QoS requirements, thus it efficiently handles run-time task workload changes. The overhead introduced by the coordination needed is kept low. The overall approach was validated on an accurate virtual platform running real RT benchmarks and results in a performance improvement according to very well know indexes.

VIII. ACKNOWLEDGMENTS

The work described in this publication was supported by the PREDATOR Project funded by the European Community's 7th Framework Programme, Contact FP7-ICT-216008. The authors would also like to acknowledge the support of ArtistDesign European Network of Excellence.

REFERENCES

[1] M. Horowitz and W. Dally, "How scaling will change processor architecture," in *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, 15-19 2004, pp. 132 – 133 Vol.1.

[2] E. Frank, R. Wilhelm, R. Ernst, A. Sangiovanni-Vincentelli, and M. Di Natale, "Methods, tools and standards for the analysis, evaluation and design of modern automotive architectures," in *DATE '08: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2008, pp. 659–663.

[3] K. Goossens, J. Dielissen, and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, Sept.-Oct. 2005.

[4] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2004, pp. 242–247.

[5] E. Salminen, V. Lahtinen, K. Kuusilinna, and T. Hamalainen, "Overview of bus-based system-on-chip interconnections," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 2, 2002, pp. II–372 – II–375 vol.2.

[6] K. J. Lin, S. Natarajan, J. W.-S. Liu, , and T.Krauskopf, "Concord: a system of imprecise computations," *Proceedings of the IEEE Compsac*, October 1987.

[7] W.-K. Shih, "Scheduling in real-time systems to ensure graceful degradation: the imprecise computation and the deferred-deadline approaches," *Dept. of Computer Science*, October 1992.

[8] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k) firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

[9] G. Buttazzo and L. Abeni, "Adaptive workload management through elastic scheduling," *Real-Time Systems*, vol. 23, no. 1–2, pp. 7–24, July–September 2002.

[10] G. Buttazzo, M. Velasco, P. Marti, and G. Fohler, "Managing quality-of-control performance under overload conditions," in *Proc. 16th IEEE Euromicro Conference on Real-Time System*, Catania, Italy, July 2004.

[11] E. Ososanya, M. McGlone, and T. Strong, "Vlsi design of a bus arbitration module for the 68000 series of microprocessors," in *Southeastcon '94. 'Creative Technology Transfer - A Global Affair'., Proceedings of the 1994 IEEE*, 10-13 1994, pp. 398 –402.

[12] F. Petrot and D. Hommais, "A generic programmable arbiter with default master grant," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 5, 2000, pp. 749 –752 vol.5.

[13] T. D. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das, and V. Degalahal, "A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks," in *VLSID '06: Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 657–664.

[14] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 6-10 2006, p. 6 pp.

[15] 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 *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, 3-6 2007, pp. 49 –60.

[16] P. Eles, A. Doboli, P. Pop, and Z. Peng, "Scheduling with bus access optimization for distributed embedded systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 5, pp. 472 –491, oct 2000.

[17] M. Paolieri, E. Qui nones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for wcet analysis of hard real-time multicore systems," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2009, pp. 57–68.

[18] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, March 2002.

[19] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis." [Online]. Available: http://www.absint.de/aiT_WCET.pdf

[20] P. Burgio, M. Ruggiero, and L. Benini, "Simulating future automotive systems," march 2010. [Online]. Available: http://www-micrel.deis.unibo.it/ ∼burgio/pub/docs/TLMBUSTechReport.pdf

[21] *AMBA AXI Manual*, ARM Limited, 2003-04. [Online]. Available: http://www.arm.com/products/solutions/AMBA3AXI.html

[22] G. A. Clouard, G. Mastroroceo. F. Carbognani. A. Perrin. F, "Stbus communication system concepts and definitions," ST Microelectronics, Tech. Rep., 2002. [Online]. Available: http://www.st.com/stonline/products/literature/um/14178.pdf

[23] S. Microelectonics, "Stbus interconnect," ST Microelectronics, Tech. Rep., 2002. [Online]. Available: http://www.st.com/stonline/products/technologies/soc/stbus.htm

[24] *OSEK/VDX standard*, uRL: http://www.osek-vdx.org.

[25] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 40–61, January 1973.

[26] *Designing an F-14 High Angle of Attack Pitch Mode Control*. [Online]. Available: http://www.mathworks.com

[27] *CoreMark, a EEMBC banchmark*. [Online]. Available: http://www.coremark.org

[28] D. Seto, J. Lehoczky, L. Sha, and K. Shin, "On task schedulability in real-time control systems," in *Proc. 17th IEEE Real-Time Systems Symposium*, Washington DC, USA, December 1996, pp. 13–21.