Timing Effects of DDR Memory Systems in Hard Real-Time Multicore Architectures: Issues and Solutions

MARCO PAOLIERI and EDUARDO QUIÑONES, Barcelona Supercomputing Center (BSC)
FRANCISCO J. CAZORLA, Spanish National Research Council (IIIA-CSIC) and BSC

Multicore processors are an effective solution to cope with the performance requirements of real-time embedded systems due to their good performance-per-watt ratio and high performance capabilities. Unfortunately, their use in integrated architectures such as IMA or AUTOSAR is limited by the fact that multicores do not guarantee a *time composable* behavior for the applications: the WCET of a task depends on inter-task interferences introduced by other tasks running simultaneously.

This article focuses on the off-chip memory system: the hardware shared resource with the highest impact on the WCET and hence the main impediment for the use of multicores in integrated architectures. We present an analytical model that computes the worst-case delay, also known as *Upper Bound Delay* (UBD), that a memory request can suffer due to memory interferences generated by other co-running tasks. By considering the UBD in the WCET analysis, the resulting WCET estimation is independent from the other tasks, hence ensuring the time composability property and enabling the use of multicores in integrated architectures. We propose a memory controller for hard real-time multicores compliant with the analytical model that implements extra hardware features to deal with refresh operations and interferences generated by co-running non hard real-time tasks.

Categories and Subject Descriptors: J.7 [**Computer Applications**]: Computers in Other Systems—*Real time*

General Terms: Design, Performance

Additional Key Words and Phrases: Multicore, SDRAM, hard real-time, memory controller, WCET

ACM Reference Format:
Paolieri, M., Quiñones, E., and Cazorla, F. J. 2013. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. ACM Trans. Embedd. Comput. Syst. 12, 1s, Article 64 (March 2013), 26 pages.
DOI: http://dx.doi.org/10.1145/2435227.2435260

1. INTRODUCTION

In real-time Federated Architectures[Obermaisser et al. 2009] each Electronic Control Unit (ECU) is dedicated to a single function. As the number of functions implemented by Critical Real-Time Embedded systems (CRTEs) increases, so the number of ECUs does. This makes federated implementations inefficient in terms of size, weight and power

This work has been mainly funded by MERASA STREP-FP7 European Project under the grant agreement number 216415. M. Paolieri is partially supported by the Catalan Ministry for Innovation, Universities and Enterprise of the Catalan Government and European Social Funds. E. Quiñones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de 1a Cierva JCI2009-05455. This work has been partially supported by the Ministry of Science and Technology of Spain under contract TIN-2007-60625 and by the HiPEAC European Network of Excellence.
Author's address: M. Paolieri, Barcelona Supercomputing Center, C/Jordi Girona 34, 08034 Spain; email: marco.paolieri@bsc.es.

DOI: http://dx.doi.org/10.1145/2435227.2435260

consumption. Instead, Integrated Architectures, such as Integrated Modular Avionics (IMA)[ARINC 1997] or Automotive Open System Architecture (AUTOSAR)[AUTOSAR 2006], have appeared as an alternative to deal with such problems. In these architectures several functions, possibly developed by different suppliers, are safely integrated into the same ECU. In our case, each function is a carried out by a task. To do so, each component (tasks in our case) of both IMA and AUTOSAR architectures have to satisfy a key timing property: *time composability*, which dictates that the timing behavior of individual components does not change by the composition. It is important to remark that time composability is a property of the tasks, and it must be guaranteed by the system.

Time composability offers two key advantages that reduce system's costs.

(1) *System Integration*. It is well recognized that the most difficult and expensive problems, that appear during the development of a system, occur during its integration; multiple components (in our cases tasks), indeed, interact at software and hardware level. It is then essential to remove the dependencies between the different sub-systems, and, in particular, it is fundamental that individual sub-suppliers are able to perform meaningful timing analysis of their components independently on other suppliers.
(2) *Incremental Qualification*. When Incremental Qualification is achieved, it is not required to re-certify unchanged components that interact with new components brought into the system. By guaranteeing such property, components can be changed or upgraded without affecting the timing behavior of the others, hence without the need to re-analyze, re-integrate and in particular re-certify the system.

The concepts of composability and predictability have different meanings in different fields. It is out of the scope of this article to compare all these *incarnations* of these two terms. Instead, we have clarified the particular meaning of the term *time predictability* used in this article. While some authors apply time composability to the execution time of tasks meaning that applications in a composable system are completely isolated and cannot affect each other's timing behaviors, we apply time composability to WCET estimations. The main reason for this is that in CRTEs, WCET estimation is the key component which schedulability analysis is based on. Our goal is that the WCET analysis for a task does not have to be repeated when any task is added/removed into/from the workload.

Multicore processors are increasingly being considered as an effective solution to cope with current and future performance requirements of real-time embedded systems [MERASA 2007] due to their low cost and good performance per watt ratio, while maintaining a relatively simple processor design. In addition, multicore processors allow scheduling mixed criticality workloads, i.e., workloads composed of safety and non-safety critical applications, inside the same processor, maximizing the hardware utilization and so reducing cost, size, weight and power requirements.

So far, at the application level, time composability has been provided by several operating system mechanisms (e.g., time partitioning, memory management unit, etc.) that deploy temporal and spatial partitioning on single-core architectures. Unfortunately, these mechanisms are not sufficient when moving towards multicore processors. It is, in fact, not possible for the OS to prevent the inter-task interferences that occur when multiple cores share several hardware resources. For example, an avionics standard, like ARINC 653, requires isolation, and such isolation cannot be provided by OS mechanisms on multicore processors because it is not possible to avoid at software level the hardware inter-core interferences.

At task level, multicore processors cannot guarantee the time composability property due to *inter-task interferences* when accessing hardware shared resources. Inter-task

interferences appear when two or more tasks access simultaneously a shared resource,[1] and so an arbitration mechanism is required to select which task is granted access to the resource. Such arbitration may delay the execution of the other tasks. Hence, inter-task interferences make the execution time, and so the WCET of a task, dependent on the other co-running tasks, not accomplishing the time composability property.

Different shared resources have different impact on the WCET of tasks depending on how often they are accessed and their delay. This article focuses on the shared resource with the highest impact on the WCET: the off-chip memory system. An improper design of the memory system may affect system's predictability [Verghese et al. 1998] as well as performance. This effect is specially high in multicore processors, where inter-task interferences caused by the off-chip shared memory system have the most significant impact on the execution time [Burger et al. 1996; Nesbit et al. 2006]. Experiments presented in Pellizzoni et al. [2010] measured a WCET increment of 2.96 times due to memory interferences on a real multicore processor. Thus, in order to enable a safe use of multicore processors in integrated architectures, it is mandatory to consider the memory system into the WCET Analysis. If this is not the case, time composability cannot be guaranteed and hence integrated architectures cannot be used.

In order to enable the use of multicore processors in integrated architectures, our contributions are as follows.

(1) We present an analytical model, based on a memory controller configuration that implements a close-page row-buffer policy and a interleaved-bank address mapping scheme, to analyze in detail the impact of the SDRAM memory system on the WCET estimation. To do so, our analytical model computes the worst-case delay, namely *upper bound delay* (UBD), that a memory request can suffer due to memory interferences generated by other co-running tasks. The selection of the memory controller configuration is based on the analysis of six different configurations, varying the row-buffer policy and the address mapping schemes.
(2) We propose an implementation of a *Real-Time Capable Memory Controller* (RTCMC) for multicore processors. RTCMC is compliant with our analytical model, so it enables the use of multicores in integrated architectures. The RTCMC introduces two new features to reduce the overall WCET: (1) A mechanism to consider refresh operations in the WCET estimation, and (2) a mechanism to minimize the impact of memory interferences caused by non hard real-time tasks (NHRTs) over hard real-time tasks (HRTs) in a mixed criticality workload.

The UBD can be used in the WCET analysis in order to take into account memory interferences and obtain a safe and composable WCET estimation for any task. That is, the resulting WCET estimation of a task is independent of the memory behavior of the other co-running tasks because the worst-case memory interference scenario is considered. The UBD information can be used in both WCET analysis tools: measurement-based and static, requiring no changes to current WCET analysis tools, so the same tools and techniques that are used and valid for single-core processors can be used in the analysis of multicore processors. Thus, no effort by the WCET analysis tools developers is required, with a reduction in the overall costs.

Moreover, our model is based on generic timing constraints as defined in the JEDEC industrial standard [JEDEC 2008], so it can be used to compare different JEDEC-compliant DDRx SDRAM memory devices. In particular, in this article we analyze the

[1]By default, the term resource refers to *hardware resource*.

UBD of three different JEDEC-compliant 256Mb x16 DDR2-SDRAM memory devices [JEDEC 2008]: DDR2-400B, DDR2-800C and DDR2-800E.

We evaluate RTCMC using a commercial WCET analysis tool RapiTime[RapiTime 2008] and an industrial hard real-time application, the collision avoidance algorithm provided by Honeywell (called CA application throughout the article), as well as with EEMBC benchmarks. We integrate RTCMC into a time composable multicore processor [Ungerer et al. 2010] preserving the time composability property in order to deploy it into integrated architectures. Our results, obtained using a cycle accurate simulator, show that the WCET estimation for CA obtained using RTCMC is less than 29% and 23% higher than the Maximum Observed Execution Time (MOET) when running in a memory intensive workload using DDR2-400B and DDR2-800C devices respectively.

2. BACKGROUND

2.1. An Analyzable Multicore Architecture

In order to provide safe WCET stimations, in this work we use the analyzable multicore architecture presented in Paolieri et al. [2009]. Such architecture has the following characteristics.

(1) An in-order core architecture that does not introduce timing anomalies [Lundqvist and Stenström 1999] or domino effects, hence enabling composability [Cullmann et al. 2010]. In particular, in our core design we lack any resource that allows dynamic resource allocation decisions [Wenzel et al. 2005].
(2) To deal with inter-task interferences in on-chip resources the architecture is designed to guarantee that the maximum delay a HRT can suffer due to on-chip inter-task interferences accessing the internal bus and the dynamically shared cache is upper bounded. On-chip interferences are considered in the WCET estimation by using the WCET computation mode [Paolieri et al. 2009].

In Paolieri et al. [2009], we assumed an idealized memory system where each core has its own private memory controller, so that threads did not suffer interferences in the memory controller. Chip bandwidth (pins) are one of the most costly resources that have to be minimized, so in real chip designs the HRTs have to share those pins originating inter-thread memory interferences. Therefore, it is required to take into account the memory controller to design a completely analyzable multicore architecture.

With the proposals presented in this article, we complete the design of a multicore processor, in which all hardware resources are accounted to provide a composable WCET estimation. Hence, enabling the use of multicores in Integrated Architectures. Figure 1(a) shows the multicore architecture we used for this work.

Regarding the task model, in this article we assume the following: each application is composed of a single task, i.e., no application is multi-threaded. The different tasks are independent of each other and run as part of multi-programmed workloads. Each task is executed until completion. No task preemption is allowed in order to avoid that the new task change the state of the preempted task's cache making the WCET analysis really difficult. A proposal to support task preemption in instruction caches has been done in Staschulat and Ernst [2004].

2.2. DDRx-SDRAM Fundamentals

In this article we focus on DDRx SDRAM off-chip memories[2] [Jacob et al. 2008] (DDR, DRR2 and DDR3) that are compliant with the JEDEC industrial standard [JEDEC 2008]. This type of memories is commonly used in high performance computer

[2]By default, the term DRAM memory refers to JEDEC-compliant DDRx SDRAM memory.

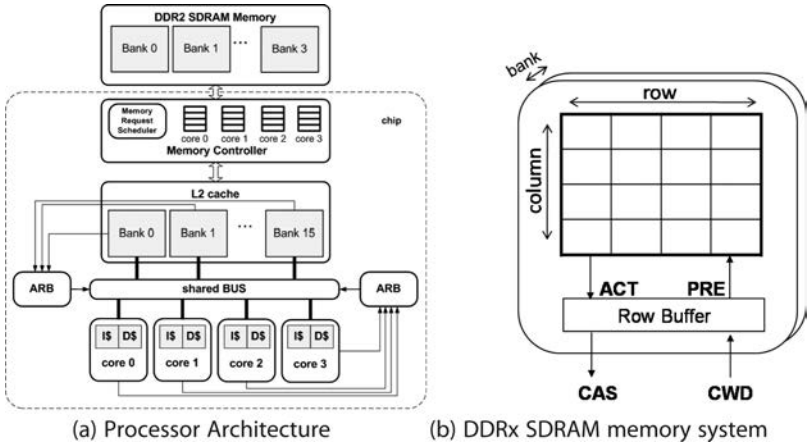

Fig. 1. Our multicore architecture and memory system.

systems, especially multicores, but designers start to introduce them into hard real-time embedded systems as performance requirements increase [Wolf 2007], e.g., both LEON3 processor by Gaisler used in space applications, and MPC8641D processor by Freescale employed in avionics industry, implement a DDR SDRAM memory controller.

A DDRx SDRAM memory system is composed of a *memory controller* and one or more *memory devices*. The memory controller is the component that controls the off-chip memory system acting as the interface between the processor and memory devices, which store the data.

A memory device consists of multiple *banks* that can be accessed independently. Each bank comprises *rows* and *columns* of DRAM cells and a *row-buffer* which caches the most recently accessed row in the bank (see Figure 1(b)). A row is loaded into the row-buffer using a row activate command (*ACT*). Such command *opens* the row, by moving the data from the DRAM array cells to the row-buffer sense amplifiers. Once a row is open, any read/write operation (*CAS*/*CWD* command) can be issued to read/write a *burst* (of size *BL*) of data from/to the row-buffer, with a latency of $BL/2$ memory cycles, i.e., the memory is double data rate. Finally, a precharge command (*PRE*) *closes* the row-buffer, storing the data back into the memory cells. The memory controller can use two different policies to manage the *row-buffer*: *close page* that precharges the row after every access and *open page* that leaves the row in the *row-buffer* open for possible future accesses to the same row.

The DRAM-access protocol defines strict timing constraints [Jacob et al. 2008], specified by the JEDEC industrial standard [JEDEC 2008] and provided by memory vendors, that dictate the minimum interval of time between different combinations of consecutive DRAM commands. Such timing constraints strictly depend on the memory characteristics. Table I summarizes the most relevant timing constraints of three JEDEC-compliant 256Mb x16 DDR2-SDRAM memory devices considered within this article: 400B, 800C and 800E. The values are expressed in memory cycles, except the clock rate of the memory that is represented in nanoseconds.

The time a request to a DRAM system memory takes to be execute depends on the *memory device* and the *memory controller*. In the memory device, the variability is originated by the *DRAM-access protocol*, which defines the different *timing constraints* and so the minimum interval of time between different combinations of DRAM commands of a memory request. In the memory controller, the variability depends from the *Row-Buffer Management policy*, the *Address-Mapping Scheme* and the *Arbitration policy*.

Table I. Timing Constraints of 256Mb x16 DDR2 Devices: 400B, 800C and 800E [JEDEC 2008]. Values in Memory Cycles

Time	Description	400B	800C	800E
t_{CK}	Clock rate in ns	5ns	2.5ns	2.5ns
t_{CAS}	CAS to bus transfer delay	3	4	6
t_{RCD}	ACT to CAS/CWD delay	3	4	6
t_{RP}	PRE to ACT delay	3	4	6
t_{RC}	ACT to ACT delay	11	22	24
t_{RAS}	ACT to PRE delay	8	18	18
t_{BURST}	Data bus transfer	4	4	4
t_{CWD}	CWD to bus transfer delay	2	3	5
t_{CCD}	CAS to CAS (same bank) delay	2	2	2
t_{RTP}	Last 4-bit prefetch of CAS to PRE delay	2	3	3
t_{WR}	End CWD to PRE delay	3	6	6
t_{WTR}	End CWD to CAS delay	2	3	3
t_{RRD}	ACT to ACT (different bank) delay	2	3	3
t_{RFC}	Time required to refresh a row	15	30	30
$\mathbf{t_{REFI}}$	REF to REF delay	1560	3120	3120

In SDRAM memories, for data integrity, the data must be periodically read out and restored to the full voltage level with *refresh* operations (*REF* commands). Every t_{REFI} cycles a refresh command (REF) is automatically sent to all banks, refreshing one row per bank. This operation takes t_{RFC} cycles to be completed.

3. OUR ANALYTICAL MODEL

In a multicore architecture several threads run in parallel and each can generate simultaneously different requests to memory. Under this scenario the timing behavior of a memory request is characterized by the *Request Inter-Task Delay* and the *Request Execution Time*. The former represents the delay the memory request can suffer, due to interferences with other requests generated by co-running tasks running on different cores, before getting the access granted to the memory device. The latter identifies the time a memory request takes to be completed, once it cannot suffer interferences with the other threads' requests. The goal of our analytical model is to define an Upper Bound Delay (UBD) to the *Request Inter-Task Delay*, so that by taking it into account during the WCET analysis of a HRT, the WCET estimations are independent from the rest of the co-running threads, and so they enable time composability.

First, we define the Request Inter-Task Delay, and then we describe how to bound it. The Request Inter-Task Delay depends on the following.

—Number of *scheduling-slots*[3] a thread has to wait before being granted access to the memory. The channel to access the memory is shared between all of them, and so they interfere with each other. Among the different memory requests that access simultaneously the memory, the memory controller is in charge of scheduling the next request. The arbitration policy implemented into the memory controller and the structure of the internal queues used to buffer the memory requests inside the memory controller determines how many *scheduling-slots* (rounds) a thread may need to wait to get access to such shared resource.

—Duration of each scheduling-slot: the *Issue Delay*. The *Issue Delay* is the time interval between the issue of two consecutive requests, i.e., from the instant a request is issued

[3]When using an arbitration policy, we define as *scheduling-slot* the round, or the time-slot of the arbitration phase in which a new request is selected by the arbiter.

until the next one can be issued. Moreover in this work, we define the Upper Bound for the Issue Delay, that we called Longest Issue Delay (t_{LID}).
The Issue Delay depends on the DRAM device used, on the specific timing constraints (see as a reference Table I), on the *Row-Buffer Management* policy and the *Address Mapping* scheme implemented in the memory controller. Depending on the row-buffer management policy used (i.e., open page or close page) and on the previous request, the sequence of DRAM commands that the memory controller issues to the memory device in order to serve a memory request can vary. That is, in DDRx DRAM memory systems the time a request takes to be executed may depend on the previous requests. For example, a read operation has a shorter duration if the previous request was also a read than if the previous request was a write. This significantly complicates the WCET analysis of HRTs because the duration of a memory request may depend on the other co-running tasks, and in a multicore, it is likely the scenario where the previous request of a certain request R is originated by a different thread. As a consequence, its execution time depends on the other co-running threads, which breaks the principle of time composability. The address-mapping scheme defines how a physical address is mapped to banks, rows and columns in the DRAM device. This can potentially originate a bank conflict if two subsequent requests access simultaneously the same bank.

In order to ensure time composability of tasks executed on multicores, we propose an analytical model based on generic timing constraints defined in the JEDEC industrial standard [JEDEC 2008], that can be used to compute the Upper Bound Delay (UBD) of the Request Inter-Task Delay. We also show that, by considering the UBD during the WCET analysis of a task, the resulting WCET estimation is independent from the workload, and hence the task becomes *time composable*.

3.1. Analysis of Different Memory Controller Configurations

As pointed in the previous section, the Request Inter-Task Delay depends on different configuration parameters of both the memory controller and the memory device. Making a complete analysis of all possible combinations of parameters is infeasible given the high number of combinations. We focus on a subset of those configurations, though our analysis can be easily extended to all the cases.

In particular this section compares six different memory controller configurations: two row-buffer policies (*open-page* and *close-page* row-buffer) and three address mapping schemes (*shared-bank*, *private-bank* and *interleaved-bank* address mapping schemes). The shared-bank scheme maps memory addresses as follows: a cache line is stored into a single memory bank and all tasks share all the memory banks. In the private-bank scheme (used in Lickly et al. [2008]), the address mapping scheme maps each cache line into the memory bank owned by the task and the other tasks cannot access it. In the *interleaved-bank* [Akesson et al. 2007] address mapping scheme, a cache line is split among all banks, so each memory request accesses all the banks in sequence (we consider a 4-bank DRAM device) and this way DRAM commands can be effectively pipelined. In the last scheme described, the access granularity is equal to $BL \cdot N_{\mathrm{banks}} \cdot W_{\mathrm{BUS}}$ bytes, where N_{banks} is the number of banks in the DRAM device, and W_{BUS} the bus width in bytes [Akesson et al. 2007].

For all the memory controller designs we assume that for every memory request a cache line of 64 bytes is transfered (i.e., the typical size for a second level cache line in high performance embedded processors [Wolf 2007]). Each of the six memory controller design configurations is analyzed for three DDR2 DRAM memory devices defined in the JEDEC industrial standard (400B, 800C and 800E); for a total of 18 cases.

Table II.
Longest Issue Delay for different memory controller configurations and three different jedec-compliant 256Mbx16 DDR2 SDRAM devices: 400B, 800C and 800E

	400B	800C	800E
close-page/interleaved-bank	21	23	27
close-page/private-bank	21	23	28
close-page/shared-bank	53	89	89
open-page/interleaved-bank	21	26	27
open-page/private-bank	9	11	13
open-page/shared-bank	28	35	43

Table II shows the results (expressed in memory cycles) of comparing the 6 memory controller configurations for 3 different memory devices. We observe that shared-bank and private-bank schemes benefit from using open-page policy since, once the row-buffer has been activated, a set of bursts can be transferred to read/write the complete cache line. Instead, when using close-page policy, the row-buffer is precharged after every read/write operation (i.e., reading/writing SDRAM bursts), enlarging the duration of the memory request. This is not the case for the interleaved-bank scheme that provides a lower Longest Issue Delay when using a close-page policy. The reason is the use of the auto-precharge feature, while the open-page policy requires sending explicitly a PRE command through the command bus, and so it results in a bus conflict for the 800C device.

When comparing the different address mapping schemes: private-bank reduces the Longest Issue Delay with respect to shared-bank because bank interferences that are originated due to tasks that access simultaneously the same memory bank are removed. In the private-bank scheme, in fact, the different HRTs share only the data and command bus of the DRAM device while in shared-bank all the components of a DRAM device are shared among the threads.

The Issue Delay has a direct impact on the UBD and so on the WCET of a HRT: the higher the Issue Delay that a memory request can suffer due to memory interferences is, the higher the WCET of the HRT. From a WCET point of view, the private-bank address scheme with open page policy provides the lowest Issue Delay. However, this approach does not provide enough system flexibility as it partitions statically the memory. It would be not possible for different tasks to use the memory according their requirements, and moreover this approach would require at least one memory bank per core, which clearly presents scalability issues at architectural implementation level. For example, in a n-core multicore processor each thread would receive a partition of $1/n - th$ of the total memory size, disregarding the memory requirements of each thread and hence leading to a poor utilization of the memory because the memory will be partitioned independently of the memory-footprint of the different HRTs.

For this reason, we select the configuration with the second lowest Longest Issue Delay but that guarantees flexibility: close-page row-buffer management policy and interleaved-bank address mapping scheme, for which we provide a complete analysis of the UBD. However it is worth noticing that, the different steps required to compute the UBD model of other memory controllers are very similar to the ones provided in this article and can be easily derived.

The first contribution of this article is shown in Table II: a comparison of the Longest Issue Delay for different memory controller and memory device configurations of DDRx SDRAM off-chip memories. This allows one to design hardware from the WCET point of view, and not from the performance point of view as usually the case.

3.2. Defining an Upper Bound to the Issue Delay

We define the Longest Issue Delay (t_{LID}), Upper Bound for the Issue Delay, using the generic timing constraints defined in the JEDEC standard (see Table I) for a memory system that implements a close-page row-buffer management policy and an interleaved-bank address mapping scheme.

t_{LID} is determined by: (1) the minimum time interval between two consecutive row activations of the same bank, that we define *time issue bank* (t_{IB}), and (2) the data bus serialization that imposes the duration of the data transfer associated with a request. t_{IB} is at least equal to $\mathrm{t_{RC}}$ cycles, that is the timing constraint that determines the minimum time interval between two row activations issued to the same bank. This guarantees that, before activating a given bank, the previous request, issued to such bank, has been completed. However, depending on the type of the previous unfinished request [Jacob et al. 2008], i.e., either a read or a write, t_{IB} is different, and it is defined as follows.

—When the previous request is a read: $t_{\mathrm{IBR}} = max\{(t_{\mathrm{RCD}} + max(t_{\mathrm{BURST}}, t_{\mathrm{RTP}}) + t_{\mathrm{RP}}), t_{\mathrm{RC}}\}$.
—When the previous request is a write: $t_{\mathrm{IBW}} = max\{(t_{\mathrm{RCD}} + t_{\mathrm{CWD}} + t_{\mathrm{BURST}} + t_{\mathrm{WR}} + t_{\mathrm{RP}}), t_{\mathrm{RC}}\}$.

Both t_{IBR} and t_{IBW} are defined as the maximum between $\mathrm{t_{RC}}$ and the sequence of timing constraints associated to the commands issued by the memory controller to perform respectively a read and a write operation.

The data bus serialization does not allow different banks to be simultaneously activated: at least t_{ACTB} of cycles equals to has to pass between activations, where $t_{ACTB} = max\{t_{RRD}, t_{BURST}\}$. This way $\mathrm{t_{RRD}}$ is satisfied and the data can be transfered from/to a given bank because the data bus is available, after the transmission from the previous bank is finished ($\mathrm{t_{BURST}}$). In addition to that, it is also required to take into account if two consecutive operations are or not of the same type: in case of *write after read*, requests are delayed due to both, the minimum time interval between the issue of a CWD and a CAS command and the $\mathrm{t_{WTR}}$ timing constraint. $\mathrm{t_{WTR}}$ accounts for the time that DRAM requires to allow I/O gating to overdrive the sense amplifiers before the read command can start, switching the direction of the bus. Thus, a *write after read* involves an additional delay of $\mathrm{t_{WTR}} + \mathrm{t_{CAS}}$. In case of *read after write*, requests are delayed by one extra cycle, because the duration of $\mathrm{t_{CWD}}$ is always defined as $\mathrm{t_{CAS}}$-1 cycles [JEDEC 2008] generating a shift on the data bus of one extra cycle.

To sum up, t_{LID} requires one to consider both the previous issued request and the current one, resulting in 4 different expressions.

—The *read-to-read issue latency* $t_{\mathrm{LIDRR}} = max\{t_{\mathrm{ACTB}} \cdot N_{\mathrm{banks}}, t_{\mathrm{IBR}}\}$ cycles.
—The *read-to-write issue latency* $t_{\mathrm{LIDRW}} = max\{t_{\mathrm{ACTB}} \cdot N_{\mathrm{banks}} + 1, t_{\mathrm{IBR}}\}$ cycles.
—The *write-to-write issue latency* $t_{\mathrm{LIDWW}} = max\{t_{\mathrm{ACTB}} \cdot N_{\mathrm{banks}}, t_{\mathrm{IBW}}\}$ cycles.
—The *write-to-read issue latency* $t_{\mathrm{LIDWR}} = max\{(t_{\mathrm{ACTB}} \cdot N_{\mathrm{banks}}) + t_{\mathrm{WTR}} + t_{\mathrm{CAS}}, t_{\mathrm{IBW}}\}$ cycles.

The worst possible scenario can be considered by defining $t_{\mathrm{LID}} = max\{t_{\mathrm{LIDRR}}, t_{\mathrm{LIDRW}}, t_{\mathrm{LIDWW}}, t_{\mathrm{LIDWR}}\}$.

By taking into account t_{LID} into the WCET analysis, the WCET estimations are time-independent from the requests sent to memory by the co-runner tasks, because the worst-case scenario is always considered. This is the key point to provide safe and composable WCET estimations in a multicore processor. Even though such technique may introduce some pessimism (our experiments, presented in Section 6, show less than 29% with actual real-time applications provided by Honeywell into the WCET estimations.

An example for a 4-bank DDR2-800E device [JEDEC 2008] is shown in Figure 2: in particular it shows the command bus (*cmd*), the data bus (*data*) and the bank status

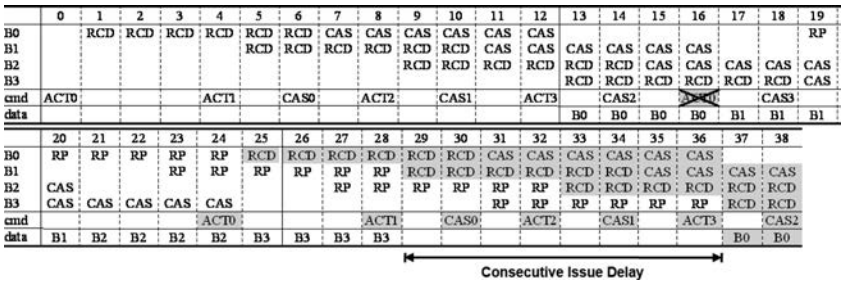

Fig. 2. Timing of two consecutive read operations (one in white and one in gray) using a 4-bank JEDEC 256Mb x16 DDR2-800E SDRAM device.

(B0 – B3) for two consecutive read requests (the first one in white and second one in grey).

In the example, each bank is activated every t_{BURST} cycles (at cycles 0, 4, 8, and 12) as in this case $t_{ACTB} = t_{BURST}$, so the data can be transferred in consecutive cycles (from cycle 13 to cycle 28). However, if a new request is ready to be served (in grey), it cannot be issued t_{BURST} cycles after activating B3 (crossed cell in cycle 16 of Figure 2) because the t_{IBR} of B0, which in this case is equal to t_{RC} cycles, would be violated. Instead, the new request must wait the *Consecutive Issue Delay* (t_{CID} cycles), being not issued until cycle 24. The t_{CID} appears when activating the first bank (B0) for the next request (or from the same request in case it preempts a NHRT request, e.g., B3-B0-B1-B2). The delay shows the gap due to the fact that the bank activation delay (e.g., t_{LIDRR} in the example) is longer than the bus transmission (e.g., t_{ACTB}*N_{banks} in the example). If t_{LIDxx} is shorter, it is hidden by the bus transmission, and so such delay becomes 0. The gap introduced in the data bus t_{CID} reduces the bus efficiency, as the bus is stalled waiting for a new bank activation. In the example above (for a read-read operation) the bus efficiency is reduced by $t_{BURST} \cdot 4/t_{LIDRR}$. Note that, if we compute t_{CID} considering DDR2-400B SDRAM device instead of DDR2-800E device, it equals 0. Hence, by using the DDR2-400B device, a new request can be issued t_{BURST} cycles after activating B3 of the previous request achieving a data bus efficiency of 100%.

High-Performance DDR2/DDR3 Memory Devices. Our model considers single-DIMM, single-rank and single 4-bank device DDRx SDRAMs, as this is currently implemented in high performance embedded systems [Jacob et al. 2008]. Instead, high-performance processors for servers, laptops, market, implement more performance-oriented techniques (e.g., request bundling), and use more complex DDR2/DDR3 DRAM memory systems in which additional timing constraints should be considered (e.g., t_{RTRS}, t_{FAW}). However, before applying any of these techniques and technologies to real-time systems, it is required to analyze the WCET impact instead of performance improvements.

Our model can be adapted to consider the additional timing constraints of this type of memories: t_{RTRS}, which determines the minimum rank-to-rank switching time, and t_{FAW}, which determines a rolling time-frame in which a maximum of four-bank activations can be engaged. t_{RTRS} should be considered in the function used to determine the delay introduced by the arbitration selection policy when two request from different ranks are sent. In case of t_{FAW}, it should be considered by the address mapping scheme in the t_{LID}.

Any additional technique to improve performance must be time analyzable to be used in real-time environments. This is out of the scope of this article, though it is part of our future work.

3.3. Refresh Operations

As already mentioned, refresh operations are one of the main contributors of the low predictability of memory systems [Bhat and Mueller 2010]. During a refresh operation no other command can be issued, hence enlarging the latency of a memory request. Since this may delay the duration of any memory request, as a consequence, it can have effects on the WCET estimations.

A possible solution to take into account this delay in the computation of the t_{LID} would be to consider that every single request is affected by a refresh operation: $t_{LID+REF} = t_{LID} + t_{RFC} - 1$. However, it is clear that applying $t_{LID+REF}$ to every memory request would lead to an overestimated WCET, because a task suffers in the worst-case only a refresh every t_{REFI} cycles. In Section 4 we show how we enhanced our real-time capable memory controller with a special hardware feature to reduce the refresh impact. In particular we synchronize the start of the execution of a HRT with the memory refresh. By doing so, the impact of refreshes is the same during both the WCET analysis and during normal execution.

3.4. Bounding the Request Inter-Task Delay: the UBD

In order to compute safe and composable WCET estimations in multicore processors, in addition to determine a safe upper bound for the Issue Delay, it is required to consider the arbitration policy and the internal request queue structure implemented in the memory controller. This can be represented by defining the function f_{LARB} that determines the maximum number of scheduling-slots (or Longest Arbitration) that a request may wait before getting the access granted, where the duration of each scheduling-slot is, in the worst case t_{LID}.

It is then possible, to define a safe upper bound for the overall Request Inter-Task Delay that Longest Request Inter-Task Delay (t_{LRITD}) or more simply the Upper Bound Delay (UBD) as a function of t_{LID}, $UBD = f_{LARB}(t_{LID})$.

In this article we consider a *round robin* arbitration policy [Paolieri et al. 2009], although other arbitration policies could be considered (like the one used in Akesson et al. [2007], TDMA, etc.). This analysis can be easily extended to other arbitration policies.

In the case of round robin, the number of inter-tasks interferences (i.e., arbitration slots) are upper bounded by the maximum number of tasks that can access simultaneously the memory (equal to the number of cores). Moreover, in order to isolate intra-task interferences (interferences originated from requests of the same task) from inter-task interferences (coming from requests of different tasks), our model considers one request queue per task. By doing this, the memory request scheduler considers only the top of each queue and not all the requests that are pending to be served from each thread. The worst-case scenario occurs when all HRTs that are executed simultaneously in the processor try to access the memory at the same time, and the HRT has to wait up to the total number of different co-running HRTs No_{HRT} (i.e., at most the total number of cores). Thus, the $f_{LARB}(x)$ function is defined as $f_{LARB\_HRT}(x) = (No_{HRT} - 1) \cdot x$ where x is the duration of each scheduling-slot, being in this case t_{LID}.

Regarding NHRTs, despite of their lower priority, it may happen that a request coming from a HRT arrives just one cycle after a request from a NHRT was issued to main memory, so the request from the HRT has to wait $f_{LARB\_NHRT}(x) = t_{LID} - 1$.

The total delay that a memory request can suffer is the sum of both effects: $UBD = f_{LARB\_HRT}(x) + f_{LARB\_NHRT}(x) = No_{HRTs} \cdot t_{LID} - 1$ where x is defined as t_{LID}.

3.5. Request Execution Time

The Request Execution Time (RET) is the amount of time a request takes to be completed (i.e., all the bits have been transfered), once it is ready and it cannot suffer

Table III.
Request Execution Time for different memory controller configurations and three different jedec-compliant 256Mbx16 DDR2 SDRAM devices: 400B, 800C and 800E

	400B	800C	800E
close-page/interleaved-bank	16	18	16
close-page/private-bank	43	70	67
close-page/shared-bank	37	70	67
open-page/interleaved-bank	16	19	17
open-page/private-bank	16	16	19
open-page/shared-bank	16	16	19

interferences with the other threads, i.e., after UBD cycles in the worst case. In particular it is the time necessary to transfer the data (until the last bit) of a memory request on the bus from/to the memory banks (starting from the first until the last bit).

In Table III we show the Request Execution Time in memory cycles for the different configurations that we explore. Request Execution Time depends on two factors: t_{ACTB} that is the interval between the activation of different banks and t_{BURST} that is the time required to transfer a burst over the data bus.

3.6. Including the Effect of the Off-chip Memory Requests into the WCET Analysis

As explained in previous sections, in a multicore processor, the timing behavior of memory requests is characterized by Request Inter-Task Delay and Request Execution Time. Therefore the maximum memory turn-around time for a memory request can be expressed as $t_{\text{mem}} = UBD + RET$.

In order to include the effect of the off-chip memory into the WCET estimation, the WCET analysis must be modified to take into account the t_{mem} computed using our analytical model. Depending on the technique used to estimate the WCET we can apply different strategies.

—*Measurement-based techniques.* It is required to implement the *WCET computation mode* [Paolieri et al. 2009] in the memory controller. When computing the WCET estimation of a HRT, the processor is set in this mode and the HRT under analysis is run in isolation. Under this mode, the memory controller artificially delays every memory request by UBD cycles, so the HRT considers the worst-case delay that each memory request can suffer due to memory interferences of the other co-running tasks. Once the WCET analysis of all the HRTs is completed the processor is set to *Standard Execution Mode* in which no artificial delay is introduced. In this work we use the measurement-based WCET analysis that follows the approach presented in Bernat et al. [2002] and it is implemented in the commercial tool RapiTime[RapiTime 2008]. RapiTime computes the WCET estimation of a program as a whole probability distribution of the execution time of the longest control-flow path, from which the absolute upper bound (i.e., the WCET estimation) is obtained. To do so, RapiTime first derives an upper bound of the Maximum Observed Execution Time (MOET) for a particular section of code (generally a basic block) from measurements; such execution times are then combined with the control flow graph to determine an overall estimation for the longest control-flow path through the program. In order to generate the control flow graph, RapiTime automatically instruments the program to be analyzed through annotations. This information is then processed by RapiTime to rebuild the actual control flow graph and to determine the execution time of the longest path based on measurements.

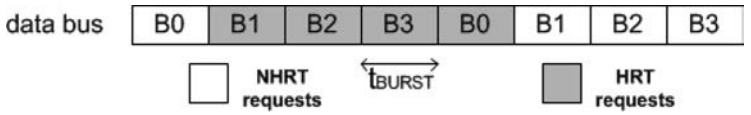

Fig. 3. Data bus utilization of 2 different memory requests. A HRT request (in gray) preempts a NHRT request (in white). For each element it is indicated from which bank (Bn) the burst is transmitted.

—*Static-based techniques*. It is required to introduce our analytical model into the model used for the processor. In particular, instead of considering a fixed latency, whenever accessing main memory the processor model should consider that each memory request requires t_{mem} to be completed: UBD cycles to be issued (i.e., the worst case needs to be considered to provide a safe WCET) and RET cycles to be executed.

4. THE REAL-TIME CAPABLE MEMORY CONTROLLER

This section presents our Real-Time Capable Memory Controller (RTCMC): a memory controller designed from a WCET point of view, and not from the performance point of view as it is generally the case in embedded and high performance systems. RTCMC is based on our analytical model, allowing to bound the effect that memory interferences have on the WCET. To that end, it implements a close-page management policy, an interleaved-bank address mapping scheme and a round robin arbitration policy with private request queue per core. RTCMC also implements special hardware features to deal with memory refresh operations and it minimizes the effect of NHRTs over HRTs, reducing the WCET estimation of HRTs; requests from HRTs are, in fact, allowed to preempt in-flight memory requests from NHRTs at memory bank boundaries.

4.1. Reducing the Effect of NHRTs on HRTs

This section presents a new hardware mechanism that allows memory requests from HRTs to preempt in-flight NHRTs memory requests, reducing the memory interference caused by NHRTs on HRTs. Our technique exploits the independent functionality of the different banks allowing preemption of different requests on bank-boundaries. In the past a mechanism to preempt on-flight requests has been described in Lee et al. [2005]. In particular, in the interleaved-bank address-mapping scheme, every memory request is composed of a sequence of independent accesses to all memory banks, so before issuing a NHRT request to a bank, our memory controller checks if there is any pending HRT request. If this is the case, the scheduler will prioritize the HRT request, stopping the issue of the remaining banks of the NHRT and assigning those issue slots to the HRT.

For example, let us assume that a request from a NHRT accesses bank B0 but before accessing B1, a request from a HRT arrives. In this scenario, RTCMC stops the NHRT request and gives the issue slot to access B1 to the HRT request. Once the HRT request is completed (sequence B1,B2, B3 and B0), the NHRT request is resumed starting from the bank it was preempted, i.e., B1. This example is shown in Figure 3. It is important to remark that our technique always maintains the same bank access order (for this article banks B0, B1, B2, and B3), exploiting the benefits of the multi-bank memory devices.

The use of this technique requires to re-define our analytical model; concretely $f_{LARB\_NHRT}(x)$. It is in fact, not required to wait until the whole NHRT request is finished ($t_{LID} - 1$ cycles, in our case) but it is required to wait until the activation of the next bank. Thus, the worst-case scenario occurs when a HRT arrives one cycle after the NHRT request has activated the last bank (B3), as it has the longest bank-to-bank activation interval. In that case the HRT request must wait $t_{\text{ACTB}} + t_{\text{CID}}$ cycles to activate

first bank B0 (see Figure 2), resulting in: $f_{LARB\_NHRT}(x) = t_{ACTB} + t_{CID} - 1$. Therefore, the new UBD that a HRT can suffer is: $UBD = (No_{\text{HRTs}} - 1) \cdot t_{LID} + t_{\text{ACTB}} + t_{CID} - 1$.

Note that the new $f_{LARB_{NHRT}}$ does not depend on t_{LID} anymore. Moreover in certain DRAM devices like DDR2-400B, the $f_{LARB\_NHRT}(x)$ becomes equal to $t_{ACTB} - 1$ cycles because t_{CID} is 0.

Our technique reduces the impact of NHRTs on HRTs from 17 cycles to 3 cycles in case of DDR2-400B, from 22 to 10 cycles in case of DDR2-800C and from 26 to 14 cycles in case of DDR2-800E, representing a reduction of 82%, 55% and 40% respectively.

4.2. Refresh Operations

As pointed out in Section 3, refresh operations can delay HRT memory requests, and this may increase the WCET of a task. To consider the delay introduced by refresh operations, a possible solution would be to account for such delay in every memory request. However, such an approach involves a huge overestimation of the WCET, because a task suffers exactly N_{REF} refresh interferences and not one for every memory request.

The number of refresh operations occurring during the WCET of a HRT is defined by the following iteration: $N_{REF}^{k+1} = \lceil (WCET_{\text{NOREF}} + N_{REF}^{k} \cdot t_{RFC}) \div t_{REFI} \rceil$ where $WCET_{\text{NOREF}}$ is the WCET estimation without considering the impact of refreshes and N_{REF} is the total number of refreshes.

Theoretically it is possible to add the delay suffered due to memory refreshes on top of the estimated WCET. Thus, the new WCET could be computed as: $WCET_{\text{TOTAL}} = WCET_{\text{NOREF}} + N_{\text{REF}} \cdot t_{\text{RFC}}$. However, depending on the approach used to perform the WCET analysis, i.e., static-based or measurement-based, it would be impossible to compute $\text{WCET}_{\text{NOREF}}$. Static-based approaches are based on abstract processor models, so $\text{WCET}_{\text{NOREF}}$ could be computed. While, measurement-based approaches are based on measurements on real processors that execute the tasks, and so refresh operations cannot be disabled. In this latter scenario it is impossible to compute $\text{WCET}_{\text{NOREF}}$. Moreover, (1) the instant or the point of the program in which the refresh occurs may have a different effect on the WCET and (2) the particular time instant in which the refresh operation occurs cannot be statically determined because it depends on the exact instant in which the application starts.

To reduce such WCET overestimation, we propose an alternative solution: synchronizing the start of a HRT with the occurrence of a refresh operation during analysis and execution time, so in both cases the start of the task is delayed until the first refresh operation takes place. By doing so, refresh commands will produce the same interferences during the analysis and the execution as they will occur exactly at the same instant with respect to the start of the task. Hence, our technique only requires one to take into account the delay introduced due to the synchronization, resulting in a tight WCET. In the worst case the task is launched one cycle after the memory has completed a refresh command, and so it must wait (t_{REFI} - 1) before starting to execute. The overall WCET is: $WCET_{\text{TOTAL}} = WCET + t_{\text{REFI}} - 1$ where $WCET$ is the measured WCET estimation.

Later in this section, we explain the hardware support necessary to synchronize the start of a task with refresh operations.

4.3. Accounting for UBD in the WCET Analysis

In CRTEs, WCET estimations is the key component in which schedulability analysis is based. For this reason, we provide time composability at the level of WCET estimation for a task rather than at the level of execution time, which has been the focus of other authors in other fields [Akesson et al. 2009, 2010]. This means, that WCET estimation

for a given program, and not its execution time, is independent from the workload in which that programs runs. We cannot show that our architecture is composable by experimentation: by showing that our architecture is time composable for the set of programs used in this article does not really prove that our architecture is time composable for any program, which is our objective. Instead, we show that our architecture is composable by design, meaning that, by design, the maximum delay a request can be delay by other task requests is upper bounded by UBD, which can be used to provide composable WCET estimations.

To consider the UBD provided by our analytical model in the WCET analysis, RTCMC introduces the *WCET computation mode* [Paolieri et al. 2009].

The UBD computed by our analytical model only depends on the total number of HRTs that access the shared resources, which is bounded by the number of cores, and not on the sequence or history of memory request accesses. Thus, by using it in the WCET computation mode, RTCMC enables computing a safe and *time composable* WCET estimation that is independent from the workload. For instance, if we consider an UBD computed assuming 4 HRTs (WCET-mode 4) in a 4-core processor, the WCET estimation computed for the HRT under analysis is safe for *any* workload in which the HRT runs on this processor.

One of the main advantages of RTCMC is that existing WCET analysis tools already used in single-core systems can be used to compute WCET estimations from the UBD estimations. As a matter of example, we used RapiTime (the original version unchanged) for the WCET analysis of HRTs running in our multicore architecture.

Once the WCET analysis for all the HRTs is completed, the processor is set to *Standard Execution Mode* in which no artificial delay is introduced. In Andrei et al. [2008] and Rosen et al. [2007] it has been formally proved that even if instructions execute before their estimated time in the worst case, the computed WCET is a safe upper-bound of the real WCET.

4.4. Hardware Implementation

RTCMC requires simple hardware modifications, with respect to a common memory controller, to implement the new features that make it time-analyzable and the tasks composable.

First, in order to allow HRTs to preempt requests from NHRTs, the memory controller implements different request queues: one per core for HRTs and a common one for all NHRTs, so NHRTs can save the data already transmitted before being preempted without the need of re-starting the request. Moreover, since the first accessed bank for a memory request does not require to be bank B0 but any other bank, a multiplexer is required to order the transmitted data properly.

In order to consider the refresh operation in the WCET estimation, the processor starts to fetch instructions from a new task when a refresh operation finishes. In order to implement this, the fetch unit of each core is enhanced with the following hardware mechanism: once the OS schedules a new program onto a core, the fetch unit does not start fetching instructions until a signal coming from the memory controller is high. Such signal is controlled by the memory controller, and it is enabled every time the refresh operation ends, while is low otherwise. This mechanism ensures the synchronization of the execution of an application with refresh operations.

The WCET computation mode only requires a small lookup table to store the precomputed UBD values of the different modes and a countdown counter to store the appropriate UBD that it is decremented every clock cycle. When the counter reaches zero, i.e., after UBD cycles, the memory request scheduler issues the memory request that has been artificially delayed as it would be in the worst case.

5. EXPERIMENTAL SETUP

We used an in-house cycle-accurate, execution-driven simulator compatible with Tricore ISA and derived from CarCore [Uhrig et al. 2005]. Our simulator models a multicore architecture composed of 4-cores with the following characteristics: Each core implements an in-order dual-issue pipeline with a fetch and pre-issue bandwidth of 8 and 4 instructions respectively. The size of the instruction buffer is 16, while the instruction window is 8. No branch prediction is used. Store operations do not block the pipeline and they access the cache directly, unless the write buffer is full. Each core has a private L1 cache with separated data and instruction caches. The Instruction and Data L1 caches are 8KB each (4-way, 1-bank, 8-byte per line, 1 cycle access, write-through write not-allocate policy). The L2 cache is shared among all cores and it is 128KB (16-way, 16-banks, 64-byte per line, 4 cycles access, write-back write-allocate policy). The L2 cache is organized in banks and in order to avoid storage interferences tasks are allowed to use only a subset of banks. This is enforced through a mechanism [Paolieri et al. 2009] that allows one to dynamically change task to L2 bank mapping. The latency of an L1 miss hitting in L2 is 9. Each core's private data and instruction L1 cache are connected to the dynamically partitioned L2 shared cache through an interference-aware, full-duplex shared bus [Paolieri et al. 2009]. Two bus arbiters, control the access to the bus: one handles the requests from the cores to L2 and the other handles the requests from L2 to the cores. The bus interferences are considered in the WCET estimation using the WCET computation mode technique.

Our simulation framework models a close-page/interleaved-bank memory controller, in particular RTCMC (see Section 4), interfaced with three different JEDEC-compliant 256Mb x16 DDR2 SDRAM devices: DDR2-400B, DDR2-800C and DDR2-800E, each composed of a single DIMM, single rank and a single 4-banks memory device (see Table I). We assume a CPU frequency of 800MHz, being a CPU-SDRAM clock ratio of 2 in case of DDR2-800C and DDR2-800E and 4 in case of DDR2-400B.

We model the DRAM memory system with DRAMsim [Wang et al. 2005], which we integrated inside our simulation framework. DRAMsim is a well known C-based memory system simulator, highly-configurable and parameterizable that implements detailed timing models for different type of existing DRAM memory systems.

We also integrated the RapiTime commercial tool [RapiTime 2008] inside our simulation framework in order to estimate the WCET of HRTs. We use RapiTime without any modification.

5.1. Benchmarks

We use a real hard real-time application: a Collision Avoidance (CA) algorithm provided by Honeywell Corporation that requires high-performance with high-data rate throughput. It is based on an algorithm for 3D path planning used in autonomous driven vehicles. As HRTs we also use EEMBC Autobench [Poovey 2007], a well-known benchmark suite that reflects the current real world demands of embedded systems. Moreover, in order to be representative of future hard-real time application requirements [MERASA 2007], we have increased their memory requirements without modifying the source code. To analyze the results based on the different memory requirements of each HRT, we have classified the EEMBC benchmarks into three different groups: memory-intensive demanding group (*High*) formed by *aifftr01,aiifft01* and *cacheb01*, the medium memory-intensive demanding group (*Medium*) formed by *aifirf01, iirflt01, matrix01* and *pntrch01*; and finally non memory-intensive demanding group (*Low*) formed by *a2time01, basefp01, bitmnp01, canrdr01, idctrn01, puwmod01, rspeed01, tblook01* and *ttsprk01*.

As NHRTs we use benchmarks from MediaBench II, SPEC CPU2006 and MiBench Automotive benchmark suites, which compete against HRTs for processor's shared resources. From MediaBench II, which is representative of multimedia applications, we use *mpeg2 decoder*. From SPEC CPU2006, that is an industry standardized CPU-intensive benchmark suite, we use *bzip2*. Finally, from MiBench Automotive, that is representative of embedded control systems, we use *susan* and *qsort*.

We run several experiments with mixed application workloads (composed of HRTs and NHRTs) in order to show that hard real-time constraints of HRTs are satisfied while providing performance to NHRTs with the resources not used by HRTs. Our goal is to select memory-hungry NHRTs so that they interfere as much as possible with the HRTs. For this article, we built workloads composed of 2 HRTs running on 2 cores and 2 NHRTs running on the other 2.

In order to study the impact of memory interferences on WCET estimations, we designed a very high memory-intensive synthetic benchmark called *opponent* in which each instruction is a store that systematically misses in L2 cache (L1 cache implements a write-through policy) and so it always accesses the DRAM memory system. Thus, HRTs that are co-scheduled with such task will suffer a tremendous impact on their execution time.

6. RESULTS

This section evaluates both, our proposed analytical model, and the RTCMC. In order to evaluate the analytical model, Section 6.1 presents an use case which allows selecting the best DRAM device from a WCET point of view.

In order to evaluate the RTCMC, Sections 6.2 and 6.3 show the impact that inter-task memory interferences have on the WCET estimation, based on the UBD provided by our analytical model. We also compare the impact of the DRAM memory system with the impact of the inter-task on-chip interferences on the WCET estimation [Paolieri et al. 2009]. As main metrics we use the capability to provide tight WCET estimations to HRTs when running in a multicore processor together with NHRTs and the level of performance that NHRTs achieve with the resources not used by the HRTs (Section 6.4). Finally, in Section 6.5 we compare RTCMC with a proposal of the state of the art, Predator [Akesson et al. 2007; Akesson 2010].

6.1. Use Case for the Analytical Model: Selecting the Best DRAM Device from a WCET Point of View

Our analytical model can be used to determine the best memory device from the WCET point of view. In this section we compare DDR2-400B, DDR2-800C and DDR2-800E devices. Let us consider an *execution workload* composed of four HRTs ($No_{HRT} = 4$) running simultaneously. It is important to remark that No_{HRT} is the number of HRTs running at the same time inside the processor, which is upper bounded by the number of cores (4 in our baseline setup), and not the total number of HRTs that form the complete task set[4]. By considering the clock rate of each DRAM device (t_{CK}), our analytical model can determine which DRAM device introduces the highest delay in the WCET estimation of a HRT. Table IV shows the results of this comparison. The DDR2-400B DRAM device has the highest impact on the WCET, delaying every request by 315 ns, while the DDR2-800C has the lowest impact, delaying each request by 172.5 ns, representing a reduction of 44% with respect to DDR2-400B. The DDR2-800E delays every request by 205.5 ns.

[4]The Operating System scheduler will be in charge of selecting which are the four HRTs that run simultaneously at every instant. This is out of the scope of this article.

Table IV.
UBD in ns of DDR2-400B, 800C and 800E Memory Devices, Considering $No_{HRT} = 4$

	400B	800C	800E
t_{LID}	21	23	27
UBD	63	69	81
UBD (in ns)	**315**	**172.5**	**205.5**

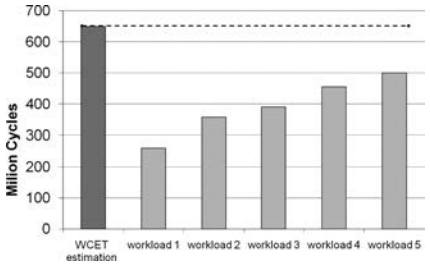

Fig. 4. WCET estimation and execution time of CA on 5 different workloads.

6.2. Composable WCET Estimations

Time composability enables Incremental Qualification in Integrated Systems, which allows upgrading or changing a given function in a system without this affecting the WCET of other functions running on the same hardware. This section evaluates the execution time variation of the CA application in the 4-core architecture that has been previously described.

Figure 4 shows the execution time of CA when it runs in 4 different workloads each composed of three more tasks chosen from the EEMBC benchmark suite. The dark grey bar shows the WCET estimation for the CA based on the UBD (as described in Section 3).

We observe that depending on the interferences the other tasks of the workload have with CA, the observed execution time is different. The difference between the Observed Execution Time of the worst workload, *workload*5 and the WCET estimation is less than 29%. Although the *workload*5 is composed of several *opponent* benchmarks, which have a very aggressive memory behavior, we cannot ensure that the *opponent* represents the worst-case scenario that CA could suffer. Building such actual worst-case opponent is hard or even impossible as it depends on the underlying processor and memory architecture and the particular HRT under study, requiring an enormous amount of time to be built and tested. Hence, a new workload composed of more memory hungry tasks, could lead to a higher execution time of CA. Any CA's execution time would be always lower than its WCET estimation, as, by design, we know which is the maximum delay CA can suffer and we take it into account in the WCET estimation.

The main conclusion we can extract is that the WCET provided is safe and time composable. At deployment time, all the other tasks with which CA runs, can be changed or updated without affecting the WCET of CA. This is a key feature of Integrated Architectures to heavily reduce design and deployment costs.

6.3. Tightness of the WCET Estimations

In this section, we further evaluate the tightness of our WCET estimations under a wide set of configurations. Figures 5 and 6 show the WCET estimation provided by RapiTime for the CA application as we vary (1) the number of HRTs running simultaneously, (2) for each HRT count, we vary the private cache partition size assigned (from

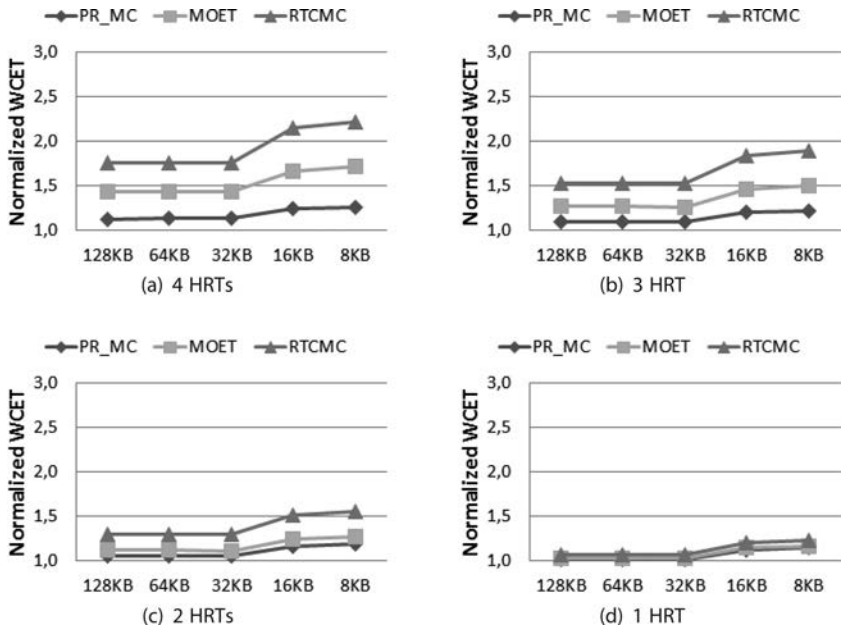

Fig. 5. Normalized WCET estimation for the CA application using JEDEC DDR2-400B.

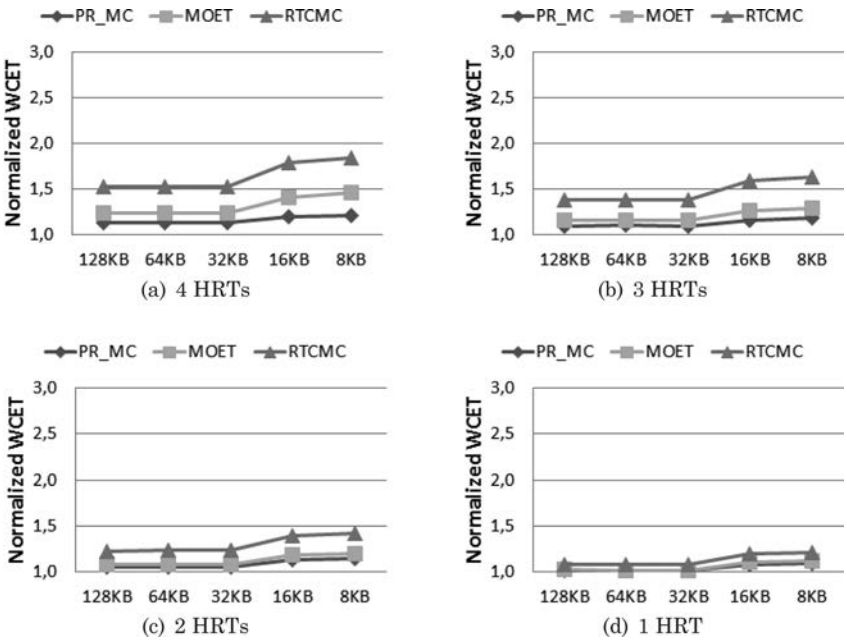

Fig. 6. Normalized WCET estimation for the CA application using JEDEC DDR2-800C.

128KB to 8KB) to show how the WCET changes as the pressure on the memory system increases. In order to ease the comparison of the WCET under different configurations, all WCET estimations are normalized to the WCET estimation in isolation, without conflicts accessing all the hardware shared resources (L2 cache, bus and DRAM memory).

For each configuration we compute the WCET estimation under two different scenarios: (1) assuming a Private DRAM Memory Controller for each HRT and so having interferences only in the on-chip resources [Paolieri et al. 2009], which has a high hardware cost in term of chip pins (labeled as *PR_MC*); and (2) on-chip and memory interferences are considered at the same time, our memory controller is shared among the different cores (labeled as *RTCMC*). For the latter scenario, in order to evaluate whether the WCET estimations are tight, we measure the Maximum Observed Execution Time (labeled as *MOET*) of the HRT when running in a high memory-intensive workload composed of several *opponents*. For example, in Figure 5(a) for the 8KB configuration we observe that when the memory controller is not shared the WCET estimation is *PR_MC* is 1.25. When the memory controller is shared and we use RTCMC the WCET estimation is around 2.22, being the MOET 1.72.

By comparing *PR_MC* and *RTCMC*, we observe that memory interferences have a tremendous impact on the WCET estimation, significantly higher than on-chip interferences. In the scenario with the biggest amount of memory accesses, i.e., assigning 8KB of L2 and 4 HRTs in the workload (Figures 6(a) and 5(a)), the memory interferences increase the WCET estimations from 1.25 (PR_MC) to 2.22 (RTCMC) using DDR2-400B and from 1.19 (PR_MC) to 1.83 (RTCMC) using DDR2-800C, which represents an increment of 0.97x and 0.64x respectively. Obviously, as memory pressure decreases, i.e., the size of the L2 cache partition assigned to the HRT increases and/or WCET computation mode decreases, the impact of memory interferences also decreases.

Even though memory interferences have high impact: By comparing the MOET and the WCET of the CA application in the highest memory-intensive workload, i.e., assigning a cache partition of 8KB and running with 3 HRTs *opponents* (Figures 6(a) and 5(a)), we observe moderate increments: 29% (from 1.72x to 2.22x) in the case of DDR2-400B and 23% (from 1.41x to 1.83x) in the case of DDR2-800C. Note that, since the actual WCET of a HRT is bigger than any MOET and equal or smaller than any WCET estimation ($MOET \leq WCET_{actual} \leq WCET_{estimation}$) [Thiele and Wilhelm 2004], the WCET estimation obtained using RTCMC is less than 29% and 23% bigger than the actual WCET for DDR2-400B and DDR2-800C respectively. The average difference in percentage between MOET and RTCMC is 22% for DDR2-400B and 20% for DDR2-800C.

Table V shows the results for the experiments described above when using as HRTs the EEMBC Automotive benchmarks in the High (H) and Low (L) groups under the worst possible scenario, running in a workload with 4 tasks. Experiments are shown for two different DRAM memory devices: DDR2-400B and DDR2-800C.

Memory interferences introduce an increment in the WCET estimation of 167% (from 2.04x to 4.84x) for the high group, 64% (from 1.51x to 2.63x) for the medium group and 15% (from 1.03x to 1.18x) using DDR2-400B. When using DDR2-800C these increments are 97%, 49% and 10% for high, medium and low groups respectively.

The WCET estimations are quite tight with respect to the MOET. For HRT counts of 2 and 4 and all cache configurations (8KB to 128kB) the WCET estimations are 37%, and 3% higher in average than the MOET for the high, and low EEMBC groups respectively, when using DDR2-400B. For the same groups the increments are 29%, and 3% respectively when using DDR2-800C. Note that, as the memory pressure decreases, i.e., less HRTs access the memory, such difference also reduces. Medium EEMBC group resembles the results of CA application.

Finally, it is important to remark that the fact of delaying every memory request by UBD cycles when computing the WCET estimation, allows our memory controller to be

Table V. Normalized WCETs for EEMBC Using DDR2-400B and DDR2-800C Respectively (H=High, M=Medium, L=Low) in 4 HRTs Workload

		DDR2-400B					DDR2-800C				
		128KB	64KB	32KB	16KB	8KB	128KB	64KB	32KB	16KB	8KB
H	PR_MC	1.27	1.40	1.46	1.67	2.04	1.27	1.34	1.38	1.49	1.67
	MOET	1.11	1.44	1.67	2.30	3.67	1.07	1.24	1.37	1.75	2.53
	RTCMC	1.35	1.94	2.25	3.10	4.84	1.32	1.65	1.83	2.32	3.29
M	PR_MC	1.18	1.18	1.32	1.47	1.51	1.18	1.18	1.26	1.35	1.37
	MOET	1.09	1.07	1.43	1.78	1.94	1.05	1.05	1.24	1.46	1.53
	RTCMC	1.26	1.28	1.82	2.44	2.63	1.24	1.25	1.57	1.93	2.04
L	PR_MC	1.02	1.02	1.03	1.03	1.03	1.02	1.02	1.02	1.03	1.03
	MOET	1.08	1.09	1.10	1.10	1.12	1.05	1.05	1.06	1.06	1.07
	RTCMC	1.12	1.13	1.15	1.15	1.18	1.09	1.09	1.11	1.11	1.13

time composable. However, the UBD can be used in non-composable systems as well, for example by applying it only to certain memory requests if information about the actual inter-task interferences that a given task can suffer is available. By doing this, the WCET estimation can be reduced at the price of breaking the time composability property as the WCET estimation will be fully dependent of the composition.

6.4. NHRT Performance Impact

The main goal of our architecture is to allow estimating tight, safe and composable WCETs for HRTs. The second goal is to get the maximum performance for the NHRTs that will benefit from the resources not used by the HRTs. In this section we analyze the IPC throughput we obtain for the NHRTs. It is computed adding the IPC of all the NHRTs. We composed workloads with 4 threads, 2 HRTs and 2 NHRTs. We use HRTs with different memory demands: *high*, *medium* and *low* groups. We generate a memory-intensive pair *mpeg2dec* - *qsort*, which frequently misses in L2; and a CPU-intensive pair *susan* - *bzip2*. We study the throughput of the NHRTs when they run simultaneously with other HRTs, normalized to the case when the NHRTs run with no other HRTs at the same time. For memory-intensive NHRTs the normalized throughput ranges from 70% when they run with HRTs from low group, up to 50% when they run with memory-intensive HRTs. CPU-intensive NHRTs do not show any performance degradation. Their normalized throughput ranges between 97% and 99% as their number of L2 misses is reduced. We observe that, our memory controller effectively allows NHRTs to exploit all the resources not used by the HRTs. As a rule of thumb, we can claim that the higher the pressure of HRTs on the memory controller is, the lower the bandwidth available for NHRTs and hence the lower the performance are.

6.5. Comparing RTCMC and Predator

This section compares RTCMC with Predator [Akesson et al. 2007; Akesson 2010]. Predator is a memory controller for multi-processor system-on-chip designed for dataflow applications (e.g., some multimedia applications). In this article instead we focus on high-integrity systems (hard real time systems). One of the main objectives of our proposal is not to require any change in current WCET static and measurement based tools used in high-integrity systems. The proposals of Akesson et al. [2007] and Akesson [2010] cannot be applied in this type of systems as there is no certified tool that provides worst-case bandwidths estimations in hard-real time environments. Although the applications of both memory controllers, RTCMC and Predator are different, in this section we provide some quantitative comparison.

Predator guarantees an user-defined bandwidth to tasks based on user-defined fixed task priority. Task priorities are defined in a strict monotonic fashion, i.e., two different

Table VI. Normalized WCET Estimation of CA Application Using UBDs Provided by RTCMC and Predator [Akesson et al. 2007]

RTCMC				PREDATOR			
	128KB	32KB	8KB		128KB	32KB	8KB
1 HRT	1.00	1.07	1.23	priority 0	1.36	1.67	1.98
2 HRTs	1.30	1.30	1.56	priority 1	1.70	2.31	2.99
3 HRTs	1.53	1.53	1.89	priority 2	2.37	3.59	5.02
4 HRTs	1.76	1.76	2.22	priority 3	4.77	7.50	11.19

tasks cannot have the same priority. Based on the assigned bandwidth and priority, Predator provides a bound on the maximum latency of a memory request. The lower the priority of a task is the higher the upper bound delay is. In fact, according to the results presented in Akesson et al. [2007] assigning a priority of 3 will involve an upper bound delay 8 times higher than assigning a priority of 0. Defining the proper bandwidth is crucial because if a thread has higher bandwidth requirements that what expected by the user, the upper bound delay of a memory request may increase, involving potential deadline misses. Predator implements the same address mapping scheme used in this article: interleaved-bank. Predator focuses on streaming/multimedia real-time applications, in which bandwidth requirements can be easily determined. However, in other application domains, such as control-based applications, bandwidth requirements are unknown, hence we target different systems. RTCMC approach requires neither knowing the bandwidth requirements nor assigning a fixed priority to each thread allowing RTCMC being applied to control-based applications. Moreover, our analytical model allows quantifying the impact of the DRAM device, being suitable to any JEDEC-compliant DRAM device: our solution defines the UBD based on the generic DRAM timing constraints and the number of HRTs running simultaneously in the processor. With RTCMC requests coming from different HRTs suffer the same UBD, however, although not presented in this article, RTCMC also allows defining priorities by applying, a two-level round-robin policy for HRTs.

In order to provide a more accurate comparison between RTCMC and Predator, we used the UBDs provided by Predator to compute the WCET estimation of the CA application using the WCET computation mode technique. Concretely, we computed 4 different UBDs, one per each priority value, using the formulas presented in Akesson et al. [2007] and considering a workload composed of four CA applications with the same bandwidth requirements, and also varying the size of the cache partition (from 128KB to 8KB) for each priority. Only DDR2-400B SDRAM device is used because that is what authors use in [Akesson et al. 2007]. Table VI shows the WCET estimation of Predator with respect to the WCET running in isolation without inter-task interferences, i.e., the same baseline of the other experiments. We observe that in the highest memory-intensive scenario, i.e., assigning a cache partition of 8KB, Predator increases the WCET estimation of the highest priority HRT by 1.98x and by 11.19x for the lowest priority HRT. Instead, by using RTCMC with the same L2 cache partition size, memory interferences increase the WCET estimation of CA application with WCET computation mode 4 only by 2.22x. Therefore, although Predator reduces the WCET estimations of the HRTs with priority 0 it increases them for HRTs with priority 1, 2 and 3 with respect to RTCMC.

7. RELATED WORK

Memory interferences have the highest impact on the WCET estimations of HRTs running on multicores, though they have not been widely studied. Many works deal with the execution of HRTs in Multicore/SMT processors [El-Haj-Mahmoud and Rotenberg 2004] [Paolieri et al. 2009][Uhrig et al. 2005][Andrei et al. 2008][Rosen et al.

2007][Hansson et al. 2009] focusing on how on-chip interferences affect the WCET analysis but without taking into account off-chip memory interferences. Predator [Akesson et al. 2007] the most similar study done in this field, is a memory controller with which we compare RTCMC in Section 6.5. However, the goal of Predator's authors is to design a memory controller real-time capable for multimedia/streaming applications and for multi-processor system-on-chip that guarantees an user-defined bandwidth to tasks based on user-defined fixed task priority. Task priorities are defined in a strict monotonic fashion, i.e., two different tasks cannot have the same priority. The PRET machine [Lickly et al. 2008] is a processor designed from a predictable point of view where the memory is statically partitioned, each thread gets different memory banks and the accesses are managed through a *memory wheel*. A TDMA-like policy is used to manage the memory wheel such that each thread can access to its bank during a static-assigned slot. However, the DRAM-access protocol is not taken into account.

In El-Haj-Mahmoud and Rotenberg [2004] it is proposed a switch-on-event single-core multi-threaded processor for real-time systems. The article presents a coarse grain timing analysis of the impact of bank and bus memory interferences on the WCET estimation without taking into account the DRAM-access protocol. Other approaches [Nesbit et al. 2006; Hur and Lin 2006; Rixner et al. 2000; Rixner 2004] focus on the DRAM-access protocol from a high performance point of view, with Quality of Service (QoS) capabilities but without upper bounds on the latency of DRAM commands so they are not real-time capable. Nesbit et al. [2006] propose a memory controller in which each task is provided with an average assigned bandwidth regardless of the load placed on the memory system from other threads.

Akesson et al. [2009] presented an approach for composable resource sharing based on latency-rate servers. As case study they describe an SRAM memory controller. Their solution, is orthogonal to ours, it would be possible to use our analytical model in combination with their arbitration scheme to design an hard real-time capable memory controller.

The impact of DRAM refresh on task execution times with the focus on how predictability is adversely affected leading to unsafe hard real-time system design is shown in Bhat and Mueller [2010]. The authors propose an approach to overcome this problem through software-initiated DRAM refresh that they tested on a single core board. With respect to our approach, their solution is software-based and so it requires a new task for the refresh to be scheduled, such task needs to preempt the execution of the current HRT. We cannot afford to have preemption because this will make the WCET analysis too complex. In contrast, our solution is hardware-based and so transparent from a software perspective.

Pitter and Schoeberl [Pitter 2007; Pitter and Schoeberl 2007; 2010] introduce a real-time Java multiprocessor called JopCMP. It is a symmetric shared-memory multiprocessor and consists of up to 8 Java Optimized Processor (JOP) cores, an arbitration control device, and a shared memory. All components are interconnected via a system-on-chip bus. No details about the memory controller and the off-chip memory are provided.

Reineke et al. [2011] propose a DRAM controller design where the DRAM device is viewed as multiple independent resources that are accessed in a periodic pipelined fashion by the PRET processor. With the proposed approach the authors eliminates contention but they force a static partitioning of the resources that does not take into consideration the memory requirements of the the different tasks.

8. CONCLUSIONS

Multicore processors have low cost and good performance per watt ratio while maintaining a relatively simple processor design. All these features make multicore

architectures very attractive for real-time system. However, such processors are highly non time composable due to inter-task interferences, making the execution time, and so the WCET of a task dependent on the other co-running threads. As a consequence, composability cannot be guaranteed, preventing a safe use of multicores in Real-Time Integrated Architectures such as IMA or AUTOSAR. In particular, the off-chip memory system is the hardware shared resource that has the highest impact on WCET due to inter-task interferences.

In this article we have proposed an analytical model that: First, it helps understanding the impact that different JEDEC-compliant DDRx SDRAM memory systems have on the WCET estimation of HRTs running in a multicore. Second, it provides an Upper Bound Delay to the maximum effect of inter-task memory interferences, enabling the computation of WCET estimations for any HRT, independently from the workload on which this task runs, and so making WCET estimations composable. Third, it allows designing hardware to reduce WCET (instead of reducing average case execution). We exemplified the use of our model by evaluating three different JEDEC-compliant 256Mbx16 DDR2 SDRAM.

Moreover, we propose a Real-Time Capable Memory Controller (RTCMC) for multicore architectures. RTCMC is compliant with our analytical model, it allows estimating the Upper Bound Delay (UBD) that each memory request can suffer, hence enabling the use of multicores in Integrated Architectures. RTCMC includes two new features: the former is a technique to deal with refresh operations, that are one of the main contributors to the variability in the low predictability of memory systems, and the latter is a mechanism to reduce the impact of NHRTs over HRTs. Overall, we have studied the problems related to inter-thread interferences accessing the main memory and proposed a possible approach to reduce the WCET of a HRT running in a multicore processor.

REFERENCES

Akesson, B. 2010. Predictable and composable system-on-chip memory controllers. Ph.D. thesis, Eindhoven University of Technology.

Akesson, B., Goossens, K., and Ringhofer, M. 2007. Predator: A predictable SDRAM memory controller. In *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*. ACM, New York, NY, 251–256.

Akesson, B., Hansson, A., and Goossens, K. 2009. Composable resource sharing based on latency-rate servers. In *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD '09)*. IEEE Computer Society, Los Alamitos, CA, 547–555.

Andrei, A., Eles, P., Peng, Z., and Rosen, J. 2008. Predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proceedings of the 21st International Conference on VLSI Design (VLSID '08)*. IEEE Computer Society, Los Alamitos, CA, 103–110.

ARINC. 1997. Specification 651: Design Guide for Integrated Modular Avionics. Aeronautical Radio, Inc.

AUTOSAR. 2006. Technical overview V2.0.1.

Bernat, G., Colin, A., and Petters, S. M. 2002. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS '02)*. IEEE Computer Society, Los Alamitos, CA, 279.

Bhat, B. and Mueller, F. 2010. Making dram refresh predictable. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS '10)*. IEEE Computer Society, Los Alamitos, CA, 145–154.

Burger, D., Goodman, J. R., and Kägi, A. 1996. Memory bandwidth limitations of future microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA '96)*. ACM, New York, NY, 78–89.

Cullmann, C., Ferdinand1, C., Gebhard, G., Grund, D., Maiza, C., Reineke, J., Triquet, B., and Wilhelm, R. 2010. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of ERTS*.

EL-HAJ-MAHMOUD, A. AND ROTENBERG, E. 2004. Safely exploiting multithreaded processors to tolerate memory latency in real-time systems. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '04)*. ACM, New York, NY, 2–13.

HANSSON, A., GOOSSENS, K., BEKOOIJ, M., AND HUISKEN, J. 2009. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst. 14,* 1, 2:1–2:24.

HUR, I. AND LIN, C. 2006. Adaptive history-based memory schedulers for modern processors. *IEEE Micro 26,* 1, 22–29.

JACOB, B., NG, S. W., AND WANG, D. T. 2008. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann.

JEDEC. 2008. DDR2 SDRAM Specification JEDEC Standard No. JESD79-2E.

LEE, K.-B., LIN, T.-C., AND JEN, C.-W. 2005. An efficient quality-aware memory controller for multimedia platform soc. *IEEE Trans. Circuits Syst. Video 15,* 5, 620–633.

LICKLY, B., LIU, I., KIM, S., PATEL, H. D., EDWARDS, S. A., AND LEE, E. A. 2008. Predictable programming on a precision timed architecture. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '08)*. ACM, New York, NY, 137–146.

LUNDQVIST, T. AND STENSTRÖM, P. 1999. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS '99)*. IEEE Computer Society, Los Alamitos, CA, 12.

MERASA. 2007. EU-FP7 project. www.merasa.org.

NESBIT, K. J., AGGARWAL, N., LAUDON, J., AND SMITH, J. E. 2006. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39)*. IEEE Computer Society, Los Alamitos, CA, 208–222.

OBERMAISSER, R., EL SALLOUM, C., HUBER, B., AND KOPETZ, H. 2009. From a federated to an integrated automotive architecture. *Trans. Comp.-Aided Des. Integ. Cir. Sys. 28,* 7, 956–965.

PAOLIERI, M., QUIÑONES, E., CAZORLA, F. J., BERNAT, G., AND VALERO, M. 2009. Hardware support for WCET analysis of hard real-time multicore systems. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, 57–68.

PELLIZZONI, R., SCHRANZHOFER, A., CHEN, J.-J., CACCAMO, M., AND THIELE, L. 2010. Worst case delay analysis for memory interference in multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*. European Design and Automation Association, Belgium, 741–746.

PITTER, C. 2007. Time predictable cpu and dma shared memory access. In *Proceedings of the International Conference on Field-Programmable Logic and its Applications (FPL '07)*.

PITTER, C. AND SCHOEBERL, M. 2007. Towards a java multiprocessor. In *Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES '07)*. ACM, New York, NY, 144–151.

PITTER, C. AND SCHOEBERL, M. 2010. A real-time java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst. 10*, 9:1–9:34.

POOVEY, J. 2007. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University.

RAPITIME. 2008. www.rapitasystems.com.

REINEKE, J., LIU, I., PATEL, H. D., KIM, S., AND LEE, E. A. 2011. Pret dram controller: Bank privatization for predictability and temporal isolation. In *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11)*. ACM, 99–108.

RIXNER, S. 2004. Memory controller optimizations for web servers. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 37)*. IEEE Computer Society, Los Alamitos, CA, 355–366.

RIXNER, S., DALLY, W. J., KAPASI, U. J., MATTSON, P., AND OWENS, J. D. 2000. Memory access scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*. ACM, New York, NY, 128–138.

ROSEN, J., ANDREI, A., ELES, P., AND PENG, Z. 2007. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07)*. IEEE Computer Society, Los Alamitos, CA, 49–60.

STASCHULAT, J. AND ERNST, R. 2004. Multiple process execution in cache related preemption delay analysis. In *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT '04)*. ACM, New York, NY, 278–286.

THIELE, L. AND WILHELM, R. 2004. Design for time-predictability. In *Design of Systems with Predictable Behaviour*.

UHRIG, S., MAIER, S., AND UNGERER, T. 2005. Toward a processor core for real-time capable autonomic systems. In *Proceedings of ISSPIT*.

Ungerer, T., Cazorla, F., Sainrat, P., Bernat, G., Petrov, Z., Rochange, C., Quinones, E., Gerdes, M., Paolieri, M., Wolf, J., Casse, H., Uhrig, S., Guliashvili, I., Houston, M., Kluge, F., Metzlaff, S., and Mische, J. 2010. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro 30*, 66–75.

Verghese, B., Gupta, A., and Rosenblum, M. 1998. Performance isolation: sharing and isolation in shared-memory multiprocessors. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*. ACM, New York, NY, 181–192.

Wang, D., Ganesh, B., Tuaycharoen, N., Baynes, K., Jaleel, A., and Jacob, B. 2005. Dramsim: a memory system simulator. *SIGARCH Comput. Archit. News 33,* 4, 100–107.

Wenzel, I., Kirner, R., Puschner, P., and Rieder, B. 2005. Principles of timing anomalies in superscalar processors. In *Proceedings of the 5th International Conference on Quality Software (QSIC '05)*. IEEE Computer Society, Los Alamitos, CA, 295–306.

Wolf, W. 2007. *High-Performance Embedded Computing*. Morgan Kaufmann.

Received November 2011; revised March 2012; accepted May 2012