A New Approach to Guarantee Critical Task Schedulability in TDMA-Based Bus Access of Multicore Architecture

Estevan Lara[1], Guilherme Debon[1], Roger Goerl[1], Paulo Villa[2], Dorian Schramm[1], Letícia B. Poehls[1], Fabian Vargas[1]
[1]*Electrical Engineering Dept., Catholic University – PUCRS*, Brazil. vargas@computer.org
[2]*Federal Institute of Rio Grande do Sul – IFRS*, Brazil. paulo.villa@eel.ufsc.br

***Abstract*— The use of multicore processors in general-purpose real-time embedded systems has experienced a huge increase in recent years. Unfortunately, critical applications are not benefiting from this type of processors as one could expect. The major obstacle is that we may not predict and provide any guarantee on real-time properties of software running on such platforms. The shared memory bus is among the most critical resources, which severely degrades the timing predictability of multicore software due to the bus access contention between cores. To counteract this problem, we present in this paper a new approach that supports mixed-criticality workload execution in a multicore processor-based embedded system. It allows any number of cores to run less-critical tasks concurrently with the critical core, which is running the critical task. The approach is based on the use of an infrastructure intellectual property (I-IP) core named Deadline Enforcement Checker (DEC) implemented in hardware, which allows the execution of any number of cores (running less-critical workloads) concurrently with the critical core (executing the critical workload). This approach allows the exploitation of the maximum performance offered by a multiprocessing system while guaranteeing critical task schedulability, i.e., that the critical task execution will not violate timing deadline. A case-study based on a dual-core version of the LEON3 softcore processor was implemented in VHDL language. Practical experiments demonstrate the proposed technique is very effective on combining system high performance with critical task schedulability within timing deadline.**

Keywords*— *Multicore processor, Critical application, High-performance embedded system, Critical task schedulability, Worst-Case Execution Time (WCET), Mixed criticality workload execution, Deadline violation.

I. INTRODUCTION

It is of common agreement among designers and users that multicore processors will be increasingly used in future embedded real-time systems for critical applications where, in addition to reliability, high-performance is at a premium. The major obstacle is that we may not predict and provide any guarantee on real-time properties of software on such platforms. As consequence, we cannot formally demonstrate that the timing deadline of critical real-time tasks cannot be violated when they are running on such platforms. In this context, the shared memory bus is among the most critical resources, which severely degrades the timing predictability of multicore workloads due to access contention among cores. In critical embedded systems (e.g., aeronautical systems) this uncertainty of the non-uniform and concurrent memory accesses prohibits the full utilization of the system performance. In more detail, the system is designed in such a way that the processor runs in the stand-alone mode when a critical task has to be executed, i.e., the bus controller allows only one core to run the critical workload, and inhibits the other cores to execute less-critical workloads till the completion of the critical task.

In order to counteract this problem and properly balance reliability against performance as long as possible, we present in this paper a new approach that supports mixed-criticality workload execution by fully exploiting processor parallelism. It allows any number of cores to run less-critical tasks concurrently with the critical core, which is running the critical task. The proposed approach is not based on any multicore static timing analysis or any timing model of multicore processor parts such as pipeline, cache memory and interactions of these parts with shared memory bus. Instead, the approach is based on the use of a dedicated infrastructure intellectual property (I-IP) core named Deadline Enforcement Checker (DEC), which works as follows: when a critical task starts running, the DEC allows the execution of any number of cores (running less-critical workloads) concurrently with the critical core (executing the critical workload) till the moment when the DEC predicts that the critical workload deadline will be violated if the processor continues running all cores concurrently from that point on. At this moment, the DEC inhibits the non-critical cores to execute less-critical tasks till the completion of the critical task by the critical core. Then, it is said that the system is switched from the "shared mode" (where two or more cores are fighting for shared memory bus access) to the "stand-alone mode", where a unique (the critical) core is running.

Given the above, the proposed approach presents the following features and advantages compared to the existing techniques:

a) It minimizes the computational complexity imposed by multicore static timing analysis: it needs only to analyze interactions between pipeline and cache models of a single core when executing a given critical task. All *inter-core conflicts* generated during the execution of the critical and the various less-critical tasks caused by their non-uniform and concurrent memory bus accesses are *not* taken into account by the DEC.

978-1-7281-1756-0/19/$31.00 ©2019 IEEE

b) From the above (a) statement, one can conclude that the proposed approach does not need the development and it is not based on timing analysis models of multicore processors. Therefore, the approach is *not* based on the faithfulness of the multicore processor model to guarantee a precise workload timing prediction.

c) In contrast to the existing techniques, the proposed approach does not require any knowledge about the implementation of the less-critical workloads or even the number of workloads that will run concurrently with the critical task. In this case, the DEC is configured by taking into account two parameters:

i) the worst-case execution time (WCET) of the critical task running in the stand-alone mode in the critical core.

ii) the timing deadline of the critical task, as specified by the application requisites.

Then, the DEC monitors online the critical task execution and automatically switches the bus access from the "shared" mode to the "stand-alone" one to guarantee the maximum possible processor performance with workload schedulability. Note that if the number of less-critical tasks changes, there is no need to re-compute the timing analysis process for the critical task to guarantee workload schedulability. This condition is ensured because the DEC can switch from the "shared mode" to the "stand-alone mode" automatically, no matters is the number of less-critical tasks running in parallel with the critical one.

d) The approach can be applied to any type of processor, considered that the designer is able to collect two signals from the processor ("*Program Counter*" and "*Annul*"). The latter signal indicates if the current instruction in the pipeline was actually executed or not. This is the case of conditional branch instructions and speculative executions, where it does not necessarily mean that instructions currently in the pipeline will be actually executed by the processor till their completion at the end of the pipe. If such instructions are flushed from the pipe, the DEC does not include them in the computation of the task execution time. So, leaving more time for the critical core to reach the task timing deadline.

e) Given that the approach can be applied to any type of processor, it allows a large spectrum of real-time operating systems to be used. Thus, traditional and well-stablished real-time operating systems for critical applications such as VxWorks [1], LynxOS, Integrity or RTEMS and their advanced versions compliant with ARINC-653 (an avionics standard for safe, partitioned systems) could also be considered in the whole system design.

In this work, the terms "task" and "workload" have the same meaning and are used interchangeably. Moreover, we assume that there is only one critical core which is in charge of running the critical task. Concurrently to this core, there can be any number of non-critical cores, each of them executing one or more less-critical tasks. The remainder of the paper is organized as follows: Section II presents the fundamentals of the problem and the existing solutions. Section III describes the proposed approach and the Deadline Enforcement Checker (DEC). Then, Section IV draws the final conclusions of the work.

II. PRELIMINARIES

A real-time computer system is a computer system where the *correctness* of the system behavior depends not only on the *logical* results of the computations, but also on the *physical time* when these results are produced. If a result has utility even after the deadline has passed, the deadline is classified as *soft*, otherwise it is *firm*. However, if severe consequences could result if a firm deadline is missed, then the deadline is called *hard* [2]. Fig. 1 depicts the basic notions concerning timing analysis of systems.

A task typically shows a certain variation of execution times depending on the input data or different behavior of the environment. The longest response time is called the *worst-case execution time* (WCET). In most cases, the state space is too large to exhaustively explore all possible executions and thereby determine the exact WCET. It is worth noting that while in the last decades WCET bound was a topic mainly related with hard real-time systems (such as aerospace and military), recently it has become crucial in other domains dealing with timing guarantees. This includes among others, the automotive industry, mobile communication and high-performance computing. In this sense, it is a mandatory condition to have an accurate determination of the WCET parameter in order to guarantee the hard-real time response of these critical systems to the environment [2].

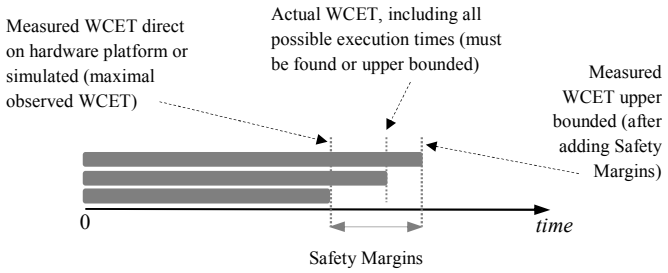

Fig. 1. Basic notions concerning timing analysis of systems.

A lot of research has been carried out within the area of WCET analysis. However, each task is, traditionally, analyzed in isolation as if it was running on a monoprocessor system. Consequently, it is assumed that memory accesses over the bus take constant amount of time to process. For multiprocessor systems with a shared communication infrastructure, however, transfer times depend on the bus load and are therefore no longer constant, causing the traditional methods to produce *incorrect* results. As response to this specific need, several approaches dealing with WCET prediction in multicore platforms have been proposed [3,4,5,6]. These approaches represent a considerable improvement of the state-of-the-art, but note:

a) They are not trivial approaches in such a way that they must not only analyze all interactions between pipeline and cache models of a single core when executing a given critical task, but they also analyze all inter-core conflicts generated during the execution of the critical and the various less-critical workloads and their non-uniform and concurrent memory bus accesses. This analysis is even more complex when the number of cores running in parallel increases.

b) If the number of tasks changes, the whole process must be recomputed in order to reschedule the tasks according to the considered bus access police (for instance, Time Division Multiple Access - TDMA, First Come First Served - FCFS or Round-Robin). For instance, in [3] authors propose a technique that predicts the critical task WCET in a TDMA-based bus police. To do so, the number of less-critical tasks must be determined. Additionally, this number must be invariable in order to precisely predict the number of TDMA slots and clock cycles must be allocated to the critical task to be completely executed. In this scenario, if the number of less-critical tasks changes, the whole process must be repeated in order to properly predict the critical task WCET, which in turn increases design and validation times.

c) It may happen that after predicting the execution of a critical task in a given multicore platform, the designer concludes that this task is not schedulable when executed in concurrence with other (less-critical) tasks. So, the whole analysis is useless and a new analysis process must restart on the basis of a smaller number of less-critical tasks to run in parallel with the critical one. The final goal is to guarantee schedulability of the critical task. If this is not attained yet, then the whole process is restarted again with an even smaller number of less-critical tasks. This "re-do" work is long and complex. So, time consuming.

d) Concerning the approach presented in [6] up to this moment, and from the best of our knowledge, it is applicable only to the Merasa processor. So, traditional processors used in embedded applications such as PowerPC, ARM and LEON3 [7] as well as well-stablished real-time operating systems for critical applications such as VxWorks [1], LynxOS, Integrity or RTEMS and their versions compliant with ARINC-653 (an avionics standard for safe, partitioned systems) cannot take advantage of this approach yet.

III. THE PROPOSED APPROACH

Fig. 2 depicts the situation where one critical task (CT) and one less critical task are running on two cores (CT is running in Core 0 and the less critical task in Core 1). When both tasks are executed (in the "Shared Mode"), the WCET of the critical task violates deadline D. Thus, the problem is unschedulable (Fig. 2a). However, if CT is executed in the "Isolated Mode", it is schedulable (Fig. 2b). In contrast with the most common used solution, where only the CT is executed at a time in the multicore platform till its completion, the proposed methodology is capable of scheduling the CT by considering the following scenario: initially both tasks are executed on the system. Then, an infrastructure intellectual property core (I-IP) named Deadline Enforcement Checker (DEC) is used to observe on-line the execution of the CT and decides switching the processor from the "Shared Mode" to the "Isolated Mode" (Fig. 2c). This switching mode condition takes place if: the distance (given in clock cycles: cc) from the "*point where the CT execution is at that moment"* to the beginning of such task is equal than the distance from the "*WCET"* to the "*Deadline D" of that task*.

Note: the point where the *CT execution is at that moment* is denoted "Critical Execution Point (CEP) of the Shared Mode". Moreover, the *end of the CT* execution is defined to be the WCET of the code executed in the Isolated Mode. See Fig. 3 for details.

This approach can be applied to any type of processor, considered that the designer is able to collect two signals from the processor ("*Program Counter*" and "*Annul*"). The latter signal indicates if the current instruction in the pipeline was actually executed or dropped down due to conditional branches or speculation-caused executions in the processor pipeline. It is worth mentioning that this approach was object of intellectual property patent deposition on Aug. 2015 [8].

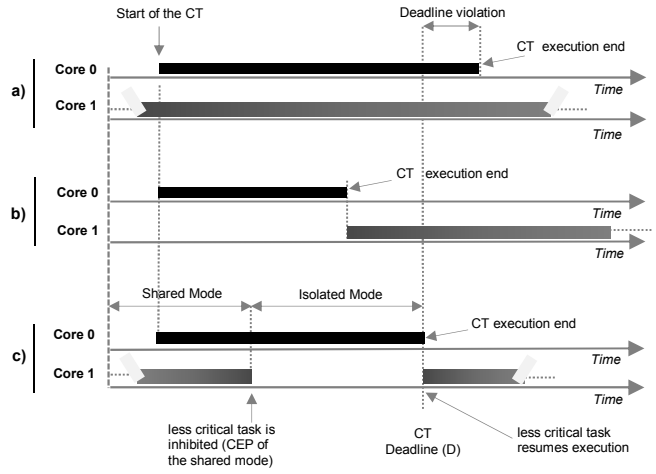

Fig. 2. Scheduling based on WCET when are considered for execution *(a)* both tasks are executed simultaneously and consequent critical task deadline violation; *(b)* basic solution: the less critical task starts executing only after critical task completion; *(c)* proposed approach.

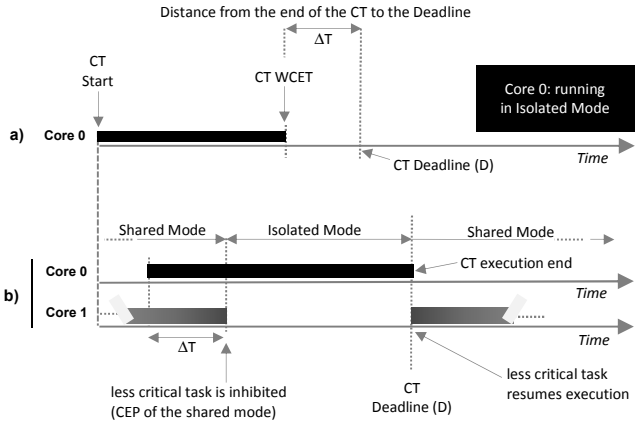

Fig. 3. Detailing the switching process between the "Shared" and the "Isolated" Modes of the proposed approach.

Therefore, the operation of the DEC I-IP is described in the following steps:

a) During CPU operation, the DEC monitors the start of the CT execution. This is performed by monitoring the address of the 1st instruction of this code stored in memory. At this moment, the DEC is operating as a "sniffer" at the instruction bus, monitoring the moment when the CPU fetches this address from the instruction memory. Before detecting the 1st instruction address of the CT (denoted by $CT_{1stInstAddr}$), the

DEC allows the system bus access police to be the Shared Mode.

b) Upon detecting the $CT_{1stInstAddr}$, the DEC initializes an internal counter with the ΔT time (the time distance, given in clock cycles, between the CT WCET and the Deadline of this code, minus a $\Delta T_{Completion}$). This counter is decremented and when it reaches zero, the DEC promotes the switching of the system bus access police from the Shared Mode to the Isolated one. Note: $\Delta T_{Completion}$ is the maximum number of clock cycles (typically, less than 10 or 12 cc) required by a given core to properly conclude the instructions that require bus access, currently under execution in its pipeline, when the bus controller switches from one core to another as ruled by the TDMA approach.

c) During the execution of the CT, the DEC monitors moment when the last instruction address ($CT_{LastInstAddr}$) of this task is fetched by the CPU. After the completion of this task, the system bus access police switches from the Isolated Mode to the Shared Mode again.

d) The DEC decrements the counter if and only if the instruction fetched by the CPU is actually executed till the end of the pipeline. Note that not all instructions fetched are actually executed by the CPU; for instance, in case of a *conditional branch*, the CPU will decide if the subsequent instructions will be executed only when such instruction reaches the execution stage of the pipeline. The same reasoning is made during *speculative execution*, where two pipes are fed with the "then" and the "else" paths of a conditional branch and then, after the execution stage, the CPU validates one pipe and flushes the other one. To follow up the above procedure, the DEC collects, at runtime, two signals from the processor ("*Program Counter*" and "*Annul*"). The latter signal indicates if the current instruction in the pipeline was actually executed or not. Simultaneously, the I-IP sniffs the address of the instructions fetched by the CPU to identify the initial and the end of the CT execution.

To perform the above operation steps, the DEC must be previously configured (during the boot of the system) with the following information:

- The WCET (given in clock cycles, cc) of the CT when running in the Isolated Mode;
- The distance (in cc) between the WCET and the Deadline of the CT minus the $\Delta T_{completion}$;
- The first and the last instruction addresses of the memory area where the CT is stored in.

A. Example of the proposed approach applied to a quad-core processor supporting TDMA-based bus access police:

Consider the following parameters for a given hypothetic application:

- Length of the TDMA Time Slice (TTS): 10,000 clock cycles (cc);
- CT WCET: 3,600,000 cc (which is equal to 360 TTS);
- CT Deadline: 4,000,000 cc (400 TTS);
- Distance between the CT WCET and the Deadline: 400,000 cc (40 TTS);
- Duration of the CT running in the critical core: 3,000.000 cc (300 TTS);
- Duration of the less critical tasks running in the less critical cores: periodical tasks, always running;
- $\Delta T_{Completion/All}$ is 1 TTS and it corresponds to the summation of $\Delta T_{Completion}$ of all cores, during the CT execution in the Shared Model.

Fig. 4 depicts the main blocks of a hypothetical *n*-core processor embedded system supporting the proposed approach.

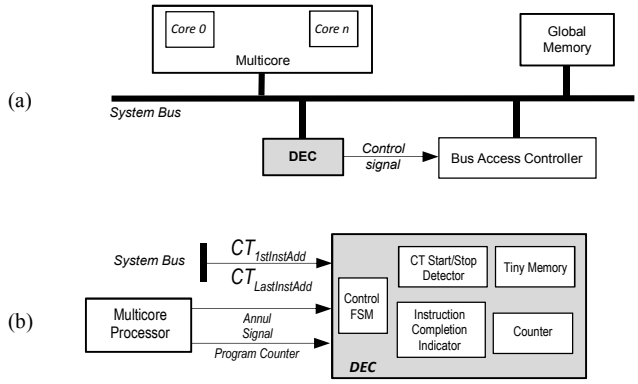

Fig. 4. Proposed approach: (a) Overview; (b) DEC general internal blocks.

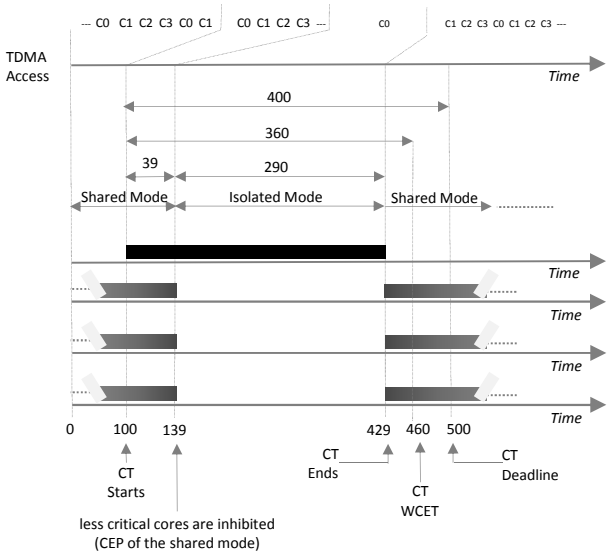

Fig. 5. Example of the proposed approach applied to a quad-core processor supporting TDMA-based bus access police.

As observed in Fig 5, the following conclusions can be taken:

1) As aforementioned, $\Delta T_{Completion/All}$ is 1 TTS. So, the number of TTS to be allocated for the system to run in the Shared Mode is: D – WCET - $\Delta T_{Completion/All}$ = 400 – 360 -1 = 39 TTS.

2) In the time interval [100 – 139] TTS, *Core0* used 10 TTS to access the system bus in the Shared Mode (out of a total of 39 TTS shared among the four cores).

3) From TTS = 139, the system runs in the Isolated Mode till the CT completion at TTS = 429.

4) Note that in order for this experiment to run properly, the DEC has to be configured with the following information:

- CT WCET: 3,600,000 cc (= 360 TTS);
- CT Deadline: 4,000,000 cc (= 400 TTS);
- $CT_{1stInstAddr}$;
- $CT_{LastInstAddr}$.

5) Note that when CT starts running (at TTS = 100) the DEC allocates automatically the immediate TDMA time slice to *Core0*, no matters was the core that was running in the preceding time slice. By doing so, the designer can precisely determines how many slices have been allocated to the CT in the Shared Mode. If such number of time slices can be determined, it is easy to compute the remaining number of time slices that is required for *Core0* to complete in the Isolated Mode (in Fig. 5, from TTS = 139 to TTS = 429) to complete the CT execution.

IV. Practical Experiment

Practical experiments have been developed in order to validate the proposed approach. With this purpose, a dual-core version of the LEON3 softcore processor was implemented in VHDL language. LEON3 [7] is the processor chosen as the target system of this work due to the significant acceptance in the scope of critical applications, in particular, aerospace. It is a synthesizable model of a 32-bit processor compatible with the SPARC V8 architecture, made available by the company Aeroflex Gaisler, under the GNU GPL license. The source code is free to use for research and educational purposes and is distributed as part of the GRLIB IP library. The whole system (processor, DEC, shared memory and applications) was run in the ModelSim Simulator, from Synopsys. Fig. 6 depicts a general overview of the implemented system.

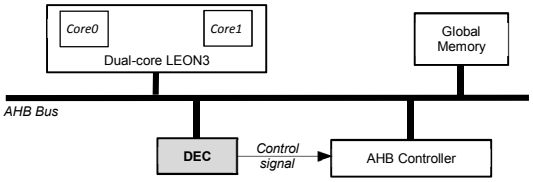

Fig. 6. Overview of the system architecture described and simulated in VHDL to validate the proposed approach.

The validation procedure is described as follows:

a) First, the whole system was simulated with the CT running in *Core0* while a less critical task was running in *Core1*, both struggling for bus access under the TDMA bus access police, without applying the proposed approach, i.e., without using the DEC I-IP.

b) Second, the same simulation was performed, but the DEC I-IP was connected to the system in order to guarantee the mixed-criticality execution of tasks running in *Core0* and *Core1*.

c) For both steps (a) and (b) above, three critical tasks were developed to run on *Core0*, one at a time: a Hamming Coder, a Bubble-Sort and a NMEA Coder. During the whole validation process, a Bubble-Sort program was running in *Core1*, as the less critical task.

Fig. 7 depicts the Control Flow Graph (CFG) of the three CTs. These tasks were selected having in mind the different complexities required to compute the WCET, from the simplest CFG (Hamming Coder) to the most complex (Bubble Sort). Hamming Coder was the easiest program to compute the WCET because it takes always the same time to run, no matter the inputs are. On the other hand, the Bubble Sort program was the most complex to compute the WCET, as can be observed in the CFG of Fig. 7c: several loops and a larger number of paths between the input and output of the CFG when compared to the NMEA coder.

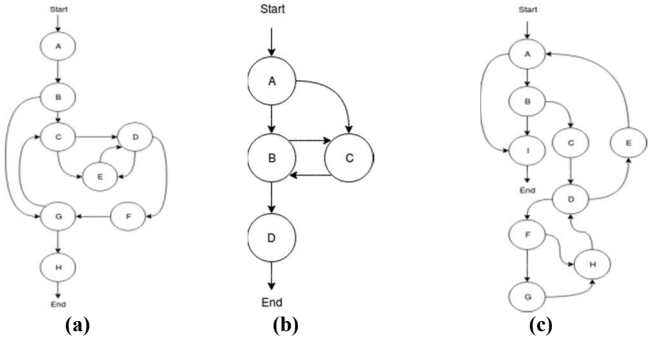

Fig. 7. Control Flow Graph (CFG) of the three CTs to run on *Core0*: **(a)** Hamming Coder; **(b)** NMEA Coder; **(c)** Bubble Sort.

Tables I, II and III summarize the following information about the three CTs aforementioned (values are given in terms of nanoseconds (ns) and clock cycles (cc) for *Core0*):

- *WCET* computed for each of the CTs;
- *Execution Time* computed for each of the CTs according to a given randomly selected input;
- *Deadline* defined for each task (for this experiment, Deadline was arbitrarily defined to be "approximately 1.35 x WCET". For instance, for the Hamming Code program the Deadline was set to 250,000 ns (or 2,000 cc) after the beginning of the program execution.

For the experiment, the length of the TDMA Time Slice (TTS) was defined as 300 cc.

TABLE I. WCET COMPUTED FOR THE THREE CTS

Critical Task (CT)	WCET Computed	
	Time (ns)	Clock Cycle (cc)
Hamming Coder	184,612.5	14,769
NMEA Coder	212,050	16,964
Bubble Sort	447,025	35,762

TABLE II. EXECUTION TIME COMPUTED FOR THE THREE CTS ACCORDING TO A GIVEN INPUT VECTOR

Critical Task (CT)	Execution Time	
	Time (ns)	Clock Cycle (cc)
Hamming Coder	147,187.5	11,775
NMEA Coder	180,312.5	14,425
Bubble Sort	427,975	34,238

TABLE III. DEADLINE ARBITRARILY DEFINED FOR THE THREE CTS

Critical Task (CT)	Deadline Defined	
	Time (ns)	Clock Cycle (cc)
Hamming Coder	250,000	20,000
NMEA Coder	300,000	24,000
Bubble Sort	600,000	48,000

Finally, Fig. 8 depicts ModelSim simulation results for the Bubble Sort program. This figure is explained as follows:

a) Signal "CT START/STOP DETECTION" (Fig. 8a): this is an internal signal generated by the DEC to indicate when it

detected the first CT instruction ($CT_{1stInstAddr}$) passing through the memory address bus towards *Core0*. At this moment, the I-IP sets up this signal to "1". Similarly, when it detects the CT last instruction ($CT_{LastInstAddr}$), it sets down this signal to "0".

b) Signal "TDMA BUS-ACCESS POLICE" (Fig. 8b): this is also an internal signal generated by the DEC to indicate to the AHB Bus Controller which is the bus access police to be applied to the system at a given moment. "01" indicates that Shared Mode should be selected from that moment on, whereas "00" indicates Isolated Mode.

c) Signal "GLOBAL MEMORY ACCESS" (Fig. 8c): this is signal used only for simulation purpose in order to observe the moments when the system memory is being accessed by *Cores 0* and *1*. When this signal is a logical "1", it means that the memory is being accessed by *Core0*, otherwise, it is being accessed by *Core1*. It is worth noting that by combining this signal with "CORE0 BUS-ACCESS REQUEST" and "CORE1 BUS-ACCESS REQUEST" one can observe that the memory access is granted in an interleaved way between the two cores.

d) Signals "CORE0 BUS-ACCESS REQUEST" and "CORE1 BUS-ACCESS REQUEST" (Fig. 8d and 8e, respectrively): they are bus-access request signals sent by *Core0* and *Core1*, respectively, to the AHB Bus Controller. These signals are active when they are equal to a logical "1". It is worth noting that during the time period when the bus is operating in the Isolated Mode ("00") the "CORE0 BUS-ACCESS REQUEST" signal is continuously switching from "0" to "1", which means that *Core0* is frequently accessing the system bus. However, "CORE1 BUS-ACCESS REQUEST" is always "1" because *Core1* is requesting bus access, but the AHB Bus Controller prevents this core from accessing the system bus during the whole period when the Isolated Mode is on.

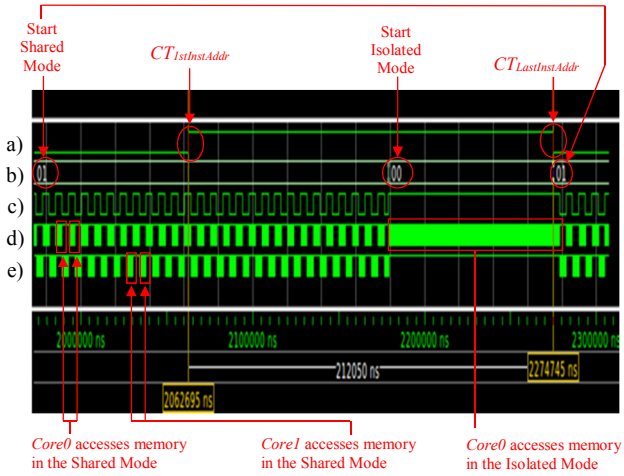

Fig. 8. ModelSim simulation results for the Bubble Sort program.

V. CONCLUSIONS

We presented a new approach that supports mixed-criticality workload execution by fully exploiting processor parallelism. It allows any number of cores to run less-critical tasks concurrently with the critical core, which is running the critical task. The proposed approach is based on the use of a dedicated infrastructure intellectual property (I-IP) core named Deadline Enforcement Checker (DEC), which works as follows: when a critical task starts running, the DEC allows the execution of any number of cores (running less-critical workloads) concurrently with the critical core (executing the critical workload) till the moment when the DEC predicts that the critical workload deadline will be violated if the processor continues running all cores concurrently from that point on. At this moment, the DEC inhibits the non-critical cores to execute less-critical tasks till the completion of the critical task by the critical core.

The DEC I-IP was described in VHDL language and coupled with the AHB bus and AHB bus controller in order to monitor the mixed-criticality execution of a dual-core version of the LEON3 softcore processor. Moreover, the AHB bus controller implemented the TDMA-based bus access police. Such approach was validated by simulating in ModelSim the whole system comprised of the processor, DEC I-IP, bus, bus controller, shared memory, three different programs representing a critical task running in the LEON3 *Core0* and a (less critical) application workload mapped to *Core1*. The experimental results demonstrated that the proposed approach is very effective on combining system high performance with critical task schedulability within timing deadline.

This approach can be applied to any type of processor, considered that the designer is able to collect two signals from the processor ("*Program Counter*" and "*Annul*"). The latter signal indicates if the current instruction in the pipeline was actually executed or not. It is worth noting that the DEC I-IP can be connected to bus controllers implementing not only the TDMA-based bus access police, but implementing any other bus access police such as First-Come First-Served (FCFS) and Round-Robin approaches.

ACKNOWLEDGMENT

This work has been supported in part by CNPq (National Science Foundation, Brazil) under contract n. 306619/2015-6 (PQ), Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) – Finance Code 001, and Federal Institute of Rio Grande do Sul (IFRS).

REFERENCES

[1] https://www.windriver.com/products/vxworks/ Last access: Nov 2018.

[2] Reinhard Wilhelm *et al.*, "The Worst-Case Execution-Time Problem – Overview of Methods and Survey of Tools", ACM Trans. On Embedded Computing Systems, vol. 7, no. 3, April 2008.

[3] Jakob Rosén, Petru Eles, Zebo Peng, Alexandru Andrei "Predictable Worst-Case Execution Time Analysis for Multiprocessor Systems-on-Chip", 2011 6th IEEE International Symposium on Electronic Design, Test and Application, pp 99-104.

[4] Sudipta Chattopadhyay, Chong Lee Kee, Abhik Roychoudhury, "A Unified WCET Analysis Framework for Multi-core Platforms", Proc. of RTAS 2012.

[5] Mingsong Lv, Wang Yi, Nan Guan, Ge Yu, Timon Kelter, Peter Marwedel, Heiko Falk, "Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software", ACM Trans. On Embedded Computing Systems, vol. 13, no. 4s, March 2014.

[6] Theo Ungerer *et al.*, "Merasa: Multicore Execution of Real-Time Applications Supporting Analyzability", IEEE Micro, Computer Society, September-October, 2010, pp. 66-75.

[7] https://www.gaisler.com/index.php/products/processors/leon3 Last access: Nov 2018.

[8] Fabian Vargas, Bruno N. Green, União Brasileira de Educação e Assistência, "Method and Device for Temporal Analysis and Control of Critical Workload in a Multicore Processor" (Método e Dispositivo para Análise e Controle Temporal da Aplicação Crítica de um Processador Multicore). Invention Patent under Register N. BR1020150191863, PCT Deposit N. 1020150191863, August 11, 2015.