Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Multicore partitioned systems based on hypervisor

A. Crespo * **M. Masmano** ** **J. Coronel** ** **S. Peiró** *
P. Balbastre * **J. Simó** *

* *Universitat Politècncia de València, Spain (e-mail: acrespo, speiro, pbalbastre, jsimo@ai2.upv.es).*
** *Fent Innovative Software Solutions S.L. (FentISS) València, Spain (e-mail: mmasmano, jcoronel@fentiss.com)*

Abstract: Multi-core processors are increasingly being considered to provide the performance required by future safety critical systems. In some domains like space, it is specially significant due to the processor technology frequency is limited by the presence of radiation. In that case, the way to increase computing power can be achieved by the use of multi-core systems.
There is a number of challenges involved in the migration to multi-core processor architectures in safety-critical embedded systems domain which are still unresolved and which contribute to increase the complexity of the design. Even if multi-core processors may offer several benefits to embedded systems, their use is not straightforward. Virtualization techniques maturity have reach the level to offer guarantees in critical systems.
In this paper, we present a multi-core hypervisor for mixed-criticality applications as one of the results of the MultiPARTES project. The paper analyse the design and implementation of XtratuM for multi-core and details a performance analysis to determine the overheads incurred by the virtualization layer and presents some results when shared resources are considered.

Keywords: Embedded control systems, Scheduling, Partitioned systems, Mixed-Criticality

1. INTRODUCTION

Control system activities have been traditionally designed as a control application with several tasks that perform the control, display data, interact with user, etc. All these activities have different level of criticality due to the timing constraints or the implication of faults. Mixed-criticality system approach tries to organize in a more coherent way the different activities according to the level of criticality they present. It has important advantages considering the robustness, fault isolation and certifiability of the system.

Mixed-criticality has increased the interest of researchers and industry for conceptualization and use where multiple components with different dependability, real-time and certification assurance levels (e.g., safety-critical and consumer functionality) are integrated into a shared computing platform Commision (2012).

On the other hand, the market for real-time embedded systems has experienced a huge growth, and it is expected to grow for the foreseeable future ARC (2012). This growth is translated in terms of increasing computing power, greater levels of security and greater performances needs. As result of this, more complex control applications will be used which can require more complex design and implementation techniques. The need of higher computing power is being covered by the adoption of multi-core architectures. Multi-cores offer better performance than single-core processors, while maintaining a relatively simple processor design. Moreover, multi-core processors ideally enable co-hosting applications with different requirements (e.g. high data processing demand and stringent time criticality). Executing non-safety and safety critical applications on a common powerful multi-core processor is of paramount importance in the embedded system market for achieve mixed-criticality systems. It allows to schedule a higher number of tasks on a single processor so that the hardware utilization is maximized, while cost, size, weight and power requirements are reduced.

There is a number of challenges involved in the migration to multi-core processor architectures in safety-critical embedded systems domain which are still unresolved and which contribute to increase the complexity of the design. Even if multi-core processors may offer several benefits to embedded systems, their use is not straightforward. On the one hand, real time embedded systems require guarantees on the timing correctness of the system, providing strong arguments for the most critical ones. On the other hand, it is required to prevent that one application could corrupt the state of other applications; paying special attention in preventing low-criticality applications to affect the high-criticality ones. This can be accomplished using time and space separation techniques; the application of such techniques to multi core processor is explored in this paper.

In order to fully exploit the performance improvements of modern processors in safety-critical applications, it is advantageous to enable the integration of applications at multiple levels of critically and security on the same processing resource. Partitioned software architectures have been designed to deal with these aspects. They have evolved to fulfill security and avionics requirements where

Copyright © 2014 IFAC

predictability is extremely important. The separation kernel proposed in Rushby (1981) established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference.

The European Space Agency (ESA) has promoted the adaptation of Integrated Modular Avionics (IMA) to cover the space market needs. The IMA-SP project ESA (2010-2012) has defined a partitioned architecture with additional services to ARINC-653 standard ARINC-653 (1996) to deal with the new challenges of future software development. Temporal and Space Partitioning (TSP) preserves the fault containment properties and development separation of concerns. The functional benefits are related to the allocation of different criticality / security classes to coexist within the same computer, management of the growth of software functionality, achieve higher degree of integration as more performing processors becomes available and facilitate design for re-use Windsor and Hjortnaes (2009).

However, applying these solutions to real-time embedded systems design is not feasible, because they could introduce timing anomalies Lundqvist and Stenström (1999) Cullmann et al. (2010) due to their non-deterministic run-time behavior.

The European Project MultiPARTES (2011) is aimed to develop a reference architecture for mixed criticality embedded systems based on virtualized open source platforms. One of the key aspects in MultiPARTES is the virtualization layer that has to achieve the requirements for mixed-criticality application development and execution. The virtualization layer is based on previous XtratuM developments adapting it for multi-core. In this paper, we describe the XtratuM Multi-core hypervisor that has been developed to support partitioned systems for real-time embedded systems. Section 2 presents the goals of the MultiPARTES project. Section 3 describes the software architecture for the virtualization layer. Section 4 details the evaluation of the hypervisor overheads and an analysis of the impact of shared resources. Finally, some conclusions are presented.

2. MULTIPARTES PROJECT

Multi-cores Partitioning for Trusted Embedded Systems (MultiPARTES) MultiPARTES (2011) is a research project supported by the European Union focused on the development of tools and solutions based on mixed criticality virtualization systems for multi-core platforms as a means to lower down development, validation and certification efforts.

The project relies on a virtualization layer to establish multiple partitions (independent execution environments) with different criticality requirements on a single hardware platform. Spatial and temporal isolation between partitions enables independent validation and certification, increases maintainability and reduces development labor.

Five engineering use cases based on existing industry examples, that have been found relevant from the methodology and platform specific demands point of view, have been identified in order to highlight the requirements of trusted embedded systems development. These use cases aim to depict current or foreseen scenarios where different levels of dependability and security are stressed, and where MultiPARTES technological innovations could achieve significant improvements on the development of future products.

2.1 Hardware Platform

The selected hardware architecture is an heterogeneous multiprocessor System-on-Chip (MPSoC) consisting of a dual-core Intel Atom processor connected to a FPGA containing up to three LEON3 CPUs via PCI-Express, where the LEON3 cores are also connected to the PCI-Express through a shared memory interface.

The Intel Atom dual-core processor is intended to accommodate general purpose operating systems running non-critical applications, and has a simple x86 architecture with local L1 and L2 cache memories and a global memory controller. The Supermicro X7SPA-H mainboard is a Commercial Off-The-Shelf (COTS) mainboard that provides expansion capabilities such as Ethernet, SATA PCI-e, USB etc. A second mainboard is supported, since one of the use cases specifically requires the use of the Intel Desktop Board D525MW. The FPGA board that will host RTOS running high criticality applications is a Xilinx Spartan6 development board that incorporates up to 3 LEON3 CPUs. Its increased logic capacity enables the use of different configurations for different use cases. The PCI-e port is used to interconnect the Atom mainboard to the FPGA board creating an heterogeneous multi-core platform.

Due to space limitations, in this paper, the analysis and performance measurements are focused on the x86 architecture. However, the results are extrapolated to the LEON3 architecture. In that case, the processor frequency is 50Mhz that impact directly in the cost of the instructions.

3. XTRATUM MULTI-CORE HYPERVISOR

3.1 SMP Virtualization layer

In order to integrate several applications on a single multi-core architecture, two software approaches can be used: the Symmetric Multi Processing (SMP) and the Asymmetric Multi Processing (AMP). Using the SMP solution a single Operating System (OS) runs on all the cores, each applications is executed in a separated process of the OS. This solution usually allows for good performance; however the development and qualification of an SMP OS is a complex and costly process. Moreover, all the applications have to use the same OS, which can be a constraint in some cases. In the AMP scheme, an independent OS is executed on each core. This obviously allows for running applications using different OS, but in this case the spatial partitioning between the applications can be difficult to enforce, as each OS has access to the complement memory map.

Instead we propose to use a hybrid solution based on a SMP virtualization layer that provides virtualization services to guest applications. Each application can run its own OS on its virtual processor. The hypervisor kernel

is able to enforce a strict time and space partitioning the of the hardware resources. Figure 1 shows the hypervisor based SMP architecture.

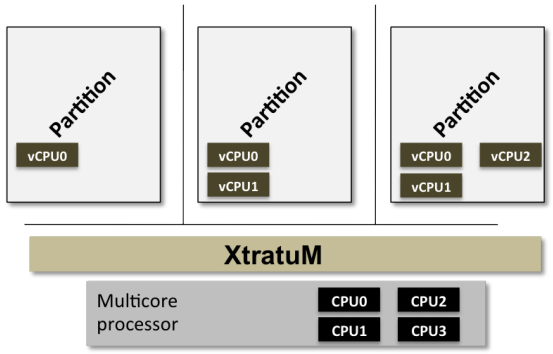

Fig. 1. Hypervisor SMP software architecture

This approach has the advantages of allowing a smooth transition from mono-core applications to multi-core platforms. An application developed to run on top of a mono-core hypervisor would run transparently on a multi-core hypervisor, only the hypervisor kernel needs to be ported and qualified to the multi-core processor. The main advanced feature of the hypervisor kernel that could allow for a more efficient use of a multi-core processor in a time and space partitioning context. The possibility to allocate several virtual processing cores to a single partition at the same time. This allows allocating more processing resource to a given partition, for instance a data processing partition.

3.2 XtratuM hypervisor

XtratuM Masmano et al. (2010, 2009) is a hypervisor for embedded real-time systems that initially was developed for x86 processor and ported to LEON2 and LEON3. XtratuM was designed to meet safety critical real-time requirements. The most relevant features are:

- Bare hypervisor.
- Employs para-virtualization techniques.
- Strong temporal isolation: fixed cyclic scheduler.
- Strong spatial isolation: all partitions are executed in processor user mode, and do not share memory.
- Fault management using a Health Monitor that is statically configured to confine the faults of the system and partitions.
- Fine grain hardware resource allocation via a configuration file.
- Robust communication mechanisms (XtratuM sampling and queuing ports).

Based and preserving the properties of the XtratuM, it has been adapted to be executed on the x86 and LEON3 multi-core in order to achieve a heterogeneous multi-core platform. The adaptation design has followed the next criteria:

- A hypervisor is a software layer that offers a virtual machine near the real one. From this point of view, the design goal is to provide as many virtual CPUs as the hardware provides.
- The hypervisor mimics the hardware behavior. The hypervisor, as the hardware does, offers one virtual CPU initialized to the partitions and they are responsible of the initialization of other virtual CPUs when needed.
- Partitions can be mono or multi-core. The hypervisor does not force to have multi core partitions (multi-core operating systems). A mono-core partition can be executed without the knowledge of the underlying multi-core hardware.
- Separation concerns among virtual and real CPUs. A partition using a virtual CPU can allocate it in any of the real CPUs.
- Real CPUs can be scheduled under different scheduling policies. The system integrator can decide the more appropriated scheduling policy for the real CPUs. Some CPUs can be scheduled under a cyclic scheduling policy and others can be scheduled under a priority-based scheduling.
- Partitions $vCPU$s are statically allocated in the configuration file.

Figure 2 shows an example of three partitions using different cores and the mapping between the virtual and real CPUs.

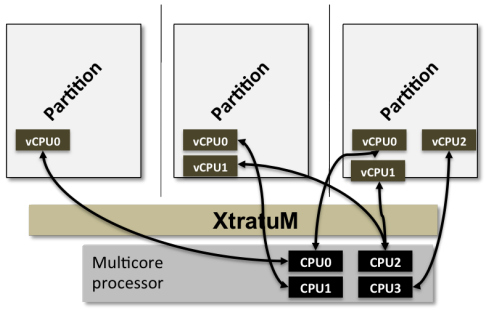

Fig. 2. virtual CPUs allocation to real CPUs

3.3 Virtual CPUs (vCPUs)

XtratuM offers as many virtual CPUs as the hardware provides. Partitions can define in the configuration file the number of $vCPU$s and its allocation in the real CPUs. XtratuM mimics the hardware behavior and boot for each partition one $vCPU$ ($vCPU0$). If more than one $vCPU$ are declared, partitions have be multi-core and are in charge of booting the rest of the declared $vCPU$s. Specific services to deal with $vCPU$s at partition level are provided by the hypervisor. These services allow the partition to start, suspend, resume and halt its $vCPU$s. $vCPU$s only are visible for the container partition.

The states of a $vCPU$ are: Normal, Halt, Suspend or Boot. For instance, a $vCPU$ of a partition can change to Boot, independently of the current state, as result of a reset on the $vCPU$ or reset partition (all $vCPU$s change to Boot). A $vCPU$ can halt itself or be halted by another $vCPU$ of the same partition. When halted, the $vCPU$ is not selected by the scheduler and the time slot allocated to it is left idle (it is not allocated to other partitions). All resources allocated to the $vCPU$ are released. It is not possible to return to normal state.

In suspended state, the $vCPU$ is not be scheduled and interrupts are not delivered. Interrupts raised while in suspended state are left pending. The $vCPU$ can return to

ready state if requested by another $vCPU$ of the partition by calling the resume $vCPU$ hypercall.

3.4 Scheduling policies

The basic scheduling policy provided by XtratuM is cyclic scheduling. This policy ensures that one partition cannot use the processor for longer than scheduled to the detriment of other partitions (temporal isolation). The set of time slots allocated to each partition is defined in the configuration file at the design phase. Each partition is scheduled for a time slot defined as a start time and a duration. Within a time slot, XtratuM allocates the processor to the partition. The cyclic schedule is defined in a Major Frame (MAF) that corresponds to the hyperperiod of the periodic activities.

However, in order to add flexibility several scheduling policies can be specified in the configuration file. The main assumptions are:

- A scheduling policy can be attached to each physical core.
- Several scheduling policies can coexist.
- All cores scheduled under cyclic scheduling share the MAF.
- Other policies are: Fixed priority scheduling.
- A multi-core partition can use several cores under different policies. These features permit to implement a fast IO communications. A multi-core partition can allocate a thread to core under a cyclic scheduling and other thread to a core under a priority based schedule.
- Real CPUs can be scheduled under different scheduling policies. The system integrator can decide the more appropriated scheduling policy for the real CPUs. Some CPUs can be scheduled under a cyclic scheduling policy and others can be scheduled under a priority-based scheduling.

4. PERFORMANCE EVALUATION

The performance evaluation is focused on the analysis of several aspects as:

- A performance model is defined to calculate the overhead introduced by the hypervisor in a partition that is executed in several slots with different durations.
- The computation of the partition context switch as the main impact of the hypervisor and the measurement of the overhead in a scenario in order to compare the computed and measured overheads.
- Effect of the hypervisor layer. The effect of the hypervisor layer is analysed by comparing the execution of the benchmarks on the native hardware against its execution as a partition on top of the hypervisor.
- Effect of the multi-core shared resources. The goal is to analyse the impact of the shared resources on the execution.

The target of evaluation is a X7SPA-H Board with Atom Dual Core at 1.66 GHz, 1 Mb L2 cache, 4Gb RAM memory DDR2 667MHz.

CoreMark benchmark have been used to perform the evaluation. CoreMark EEMBC (2001) is a simple benchmark that is designed specifically to test the functionality of a processor core. It uses basic data structures (lists, strings, and arrays) and algorithms that are common in practically any application. It allows analysing with accuracy the impact of the hypervisor layer.

The adaptation of these tests to be executed as XtratuM partitions is minimal (clock access and output).

4.1 Hypervisor layer performances

Performance tests measure the quality of the system, such as overhead or performance. The performance of the multicore hypervisor is assessed through a overhead model and a set of tests designed to capture the overheads induced by the hypervisor under different loads. Standard Coremark benchmark such as is run as bare metal applications as well as partitions, the frequency of context switches and the number of partitions executing concurrently will be increased progressively in order to measure the performance of the hypervisor at several different load points.

4.2 Overhead model

When the hypervisor schedules a partition, it is executed without impact of the hypervisor. The hypervisor is executed only if the partition explicitly request hypervisor services as set interrupt mask, send or receive a message, etc. So, the partition execution is shown in figure 3. As the partition slot is defined with a $[start\_time, duration]$ being the $start\_time$ relative to the MAF origin, when this time arrives, the hypervisor performs the partition context switch (PCS) and sets a timer to trigger at the absolute time $start\_time + duration$.

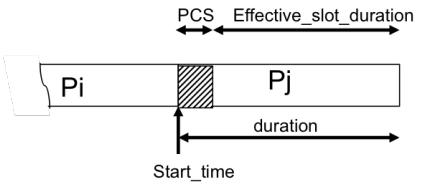

Fig. 3. Partition effective slot duration

The effective slot time used by a partition k is the time when the partition executes its own code. The effective time of a slot ET can be modeled taking into account the slot duration SD and the partition context switch PCS.

$$ET = SD - PCS$$

In general, if a partition k is executed in a MAF n times with slots of different sizes, the total effective time in the MAF can be computed as:

$$ETk = \sum_{i=1}^{n}(SD_i^k) - (n * PCS)$$

being SD_i^k the slot duration of slot i of the partition k.

The virtualization layer produces a performance loss PL in a partition k that can be modeled by:

$$PL_k = \frac{n*PCS}{\sum_{i=1}^{n} SD_i^k}$$

4.3 Partition Context Switch (PCS)

In order to measure the PCS, XtratuM has been instrumented to get the exact instant times at which the

partition switch is performed. These exact times are the clock occurrence signaling the end of the previous slot (t_1) and the return of the hypervisor code to the partition code (t_2).

These direct measurements are obtained by forcing breakpoints in the code instructions that initiate and finish the process. In debugging mode, the execution is halted each time the breakpoint is reached and the register that contains the time is logged. Measured time for the x86 processor are 11 μsec.

In order to confirm these result a scenario with several partitions is build. The scenario consists in 3 counting (increase an internal counter) partitions and a reader partition that is able to access to the partition counters. The reader partition reads the partition counters every second. Counting partitions are executed in different experiments with different slot durations. The goal of the experiment is to measure the performance loss of the counting partitions when the slot duration is 1000, 500, 100, 50, 10, 5 and 1 millisecond.

Table 1 summarizes the results comparing the theoretical PL and the measured in the experiment. The experiment with 1 second is taken as reference for the scenarios.

Slot duration (msec)	Measured overhead	Theoretical PL
1000	0,000%	0,001%
500	0,001%	0,002%
100	0,009%	0,011%
50	0,019%	0,022%
10	0,099%	0,110%
5	0,202%	0,220%
1	1,074%	1,100%

Table 1. Measured vs Calculated overhead

4.4 Native versus partition multi-core applications

This test aims to evaluate the performance loss due to the multi-core hypervisor. The goal is to compare Coremark benchmark running in the native hardware using a bare implementation against the same benchmarks running as a partition on top of the virtualization layer when it is executed in 1 core without the interference of the other core and in 2 cores at the same time intervals. The partitioned benchmarks are executed under the hypervisor cyclic scheduling. The slot duration is larger than 30 seconds in order to complete the execution in one slot and avoid in the measurement the effect of the partition context switch.

Table 2. shows the results obtained.

CoreMark	Core	Cores used	CoreMark/MHZ	Perf.loss
Native	-	-	1,78911	0,00%
Partitioned	1	1	1,78597	0,18%
Partitioned	1	2	1,77386	0,85%
Partitioned	2	2	1,77408	0,84%

Table 2. Native vs Partition performance

When comparing the native CoreMark with respect to the partitioned version, the overhead is due to the use of privileged instructions used by the benchmark that have been para-virtualized invoking hypervisor services. When the benchmark is execute at the same time in two cores an additional overhead is detected due to the use of shared resources (cache, memory, etc.).

4.5 Temporal interference due to shared resources

Temporal interference is produced when partitions in different cores use shared resources. We focus on this evaluation of the temporal impact that a target partition suffers when another partition is executed in other core and perform intensive access to memory.

To analyse this impact, a scenario with different levels of overlapping in both partitions is defined. The scenario is defined with 2 partitions. P_1 is the target of evaluation and perform a fixed payload that is measured in an isolated environment. P_1 performs the following steps: read the clock (t_1), perform the payload, read the clock (t_2) and computes the differences $t_2 - t_1$ (execution time). P_2 is a dummy partition that perform a loop accessing to a table and modifying its values. P_1 is executed in core 0 and P_2 is executed in core 1.

In order to analyse the effects in the worst conditions, cache management (instructions and data) is disabled for both partitions forcing to both partitions to access physically to memory.

This scenario is executed under the following scheduling plan:

- MAF: 50 msec
- P_1 slot duration: 20 msec. The payload is 830 μusec.
- P_2 slot duration: 20 msec.
- Experiments:
  - S0: No interference. P_2 [20ms-20msec]
  - S25: 25% of interference. P_2 [600us-20msec]
  - S50: 50 % of interference. P_2 [400us-20msec]
  - S75: 75 % of interference. P_2 [200us-20msec]
  - S100: 100 % of interference. P_2 [0us-20msec]

Fig. 4 shows the schedule of S25.

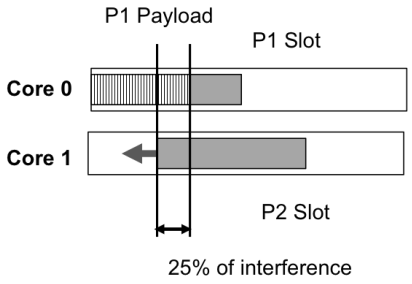

Fig. 4. Schedule of the S25

Table 3 presents the results of the impact of P_2 on the execution of P_1. If there is an approximated overlapping of the 25%, the increment of the execution time is incremented from 830 to 848 μsecs. Next table presents the statistics of these measurement that corresponds to 100 measurements in each experiment (time values are in μsecs).

Fig. 5 plots the measured values of the execution time in the different experiments. X-axis represents the measurements and Y-axis the execution time in μsecs.

These results show that the impact, as expected, depends on the overlapping interval. The important aspect is that

	S0	S25	S50	S75	S100
Avg	834	848	869	895	946
Max	837	851	872	899	948
Min	831	845	865	888	944
Stdev	1,944	1,917	2,031	2,041	1,056
Interference	-	1,68%	4,20%	7,31%	13,43%

Table 3. Impact of the memory accesses in the $P1$ execution time

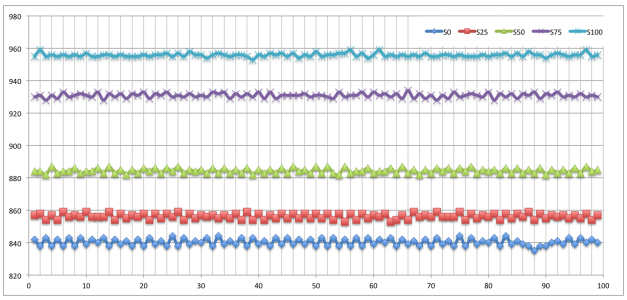

Fig. 5. *Execution time plot*

in a cyclic schedule that has been off-line generated, it can be adjusted to limit the effects of the interference. If the result in the S100 scenario (13,43 %) that corresponds to the worst case scenario is compared with the obtained in cache enabled conditions (table 4.4 0.85%), it can be conclude that still flexibility in the margins is possible.

This analysis strongly depends on the hardware platform. Specific analysis for each new platform should be performed. The analysis gives an idea about the bound of the impact. With these information, should be possible to take into account it for the worst case analysis and also to design the cyclic plan.

5. CONCLUSIONS

In this paper we presented the XtratuM hypervisor for multi-core systems developed under the MultiPARTES project to support mixed-criticality application on multi-core systems. XtratuM is a hypervisor that is being uses in space applications in single-core systems. In this work, we presented the design and implementation issues of the multi-core adaptation. Multi-cores offer better performance than single-core processors to deal with executing non-safety and safety critical applications on a common powerful multi-core processor. It is of paramount importance in the embedded system market for achieve mixed-criticality systems. It allows to schedule a higher number of tasks on a single processor so that the hardware utilization is maximized, while cost, size, weight and power requirements are reduced. Additionally, the described approach for XtratuM permits to re-use single-core applications running in a multi-core platform without impacting in the application design and implementation.

However, the use of multi-core introduce a indeterminism in the worst case execution time of the applications due to the execution in other cores. This aspect can affect to the temporal isolation of partitions because the execution time is dependent of the other core load. In the evaluation described in the paper, we shown that the effect of other cores can be estimated in the worst case. It depends on the overlapping execution that can be controlled in the cyclic scheduling plan of the hypervisor that is off-line designed and can be controlled when the plan is generated. Appropriated tools to model margins to partition worst case analysis and to generate the scheduling plan are required. In MultiPARTES project, there is a specific work package that is in charge of this activity.

Finally, the performance evaluation of XtratuM has shown the low impact on application performances. It permits to consider the virtualization approach as an important component to build future secure and robust embedded systems where several applications can have different levels of constraints. One challenge that has to be solved is the certifiability of virtualization layer for multi-core platforms. It is an important issue that has to be supported by appropriated methods and techniques.

ACKNOWLEDGEMENTS

This work has been funded by the 7th Framework Programme (FP7), project MultiPARTES (IST 287702), and by the Spanish National R&D&I program, project HI-PARTES (TIN2011- 28567-C03) and COBAMI (DPI2011-28507-C02-02).

REFERENCES

ARC (2012). Process Safety System. Market Analysis and Forecast. ARC Advisory Group.

ARINC-653 (1996). *Avionics Application Software Standard Interface (ARINC-653)*. Airlines Electronic Eng. Committee.

Commision, E. (2012). Workshop on Mixed Criticality Systems. Brussels. cordis.europa.eu/fp7/ict/computing/home en.html.

Cullmann, C., Ferdinand, C., Gebhard, G., Grund, D., (Burguire), C.M., Reineke, J., Triquet, B., and Wilhelm, R. (2010). Predictability considerations in the design of multi-core embedded systems.

EEMBC (2001). Coremark benchmark. Www.coremark.org/.

ESA (2010-2012). Integrated modular avionics for space (ima-sp). ESA project.

Lundqvist, T. and Stenström, P. (1999). Timing anomalies in dynamically scheduled microprocessors. In *IEEE Real-Time Systems Symposium*, 12–21.

Masmano, M., Ripoll, I., Crespo, A., and Metge, J. (2009). Xtratum: a hypervisor for safety critical embedded systems. In *Eleventh Real-Time Linux Workshop*. Dresden (Germany).

Masmano, M., Ripoll, I., Peiró, S., and Crespo, A. (2010). Xtratum for leon3: an open source hypervisor for high integrity systems. In *European Conference on Embedded Real Time Software and Systems. ERTS2 2010*. Toulouse (France).

MultiPARTES (2011). Multi-cores partitioning for trusted embedded systems. EU FP7-ICT-287702 2011-14.

Rushby, J. (1981). Design and verification of secure systems. In *ACM Operating Systems Review*, volume 15, 5, 12–21.

Windsor, J. and Hjortnaes, K. (2009). Time and space partitioning in spacecraft avionics. *Space Mission Challenges for Information Technology*, 0, 13–20. doi:http://doi.ieeecomputersociety.org/10.1109/SMC-IT.2009.11.