A Memory Arbitration Scheme for Mixed-Criticality Multicore Platforms

Bekim Cilku*, Alfons Crespo†, Peter Puschner*, Javier Coronel‡ and Salvador Peiro†
*Vienna University of Technology, Vienna, Austria
{bekim,peter}@vmars.tuwien.ac.at
†Universidad Politecnica de Valencia, Valencia, Spain
{acrespo,speiro}@ai2.upv.es
‡fentISS, Valencia, Spain
jcoronel@fentiss.com

***Abstract*—In mixed-criticality systems, applications of different criticality levels share the same computing platform. To avoid spatial and temporal interference of the applications, the computing platform must implement measures for spatial and temporal isolation. In this paper we show how the enhancement of a static memory arbiter by a second, dynamic arbitration layer facilitates the interference-free integration of mixed-criticality applications with different performance requirements. This paper (a) compares the performance tradeoffs of the new dual-layer arbiter and a COTS arbiter and (b) evaluates the performance of an XtratuM hypervisor system running on a platform with this dual-layer arbiter.**

I. Introduction

The high processing capability that multi-core embedded systems have reached allows us to run multiple applications on a single shared hardware platform [1]. However, some of the integrated applications may have firm real-time constraints that require a formal proof that the deadlines are met, while the others may be less demanding. For such a mixed-criticality system, only the set of critical applications needs to be certified; the rest of the applications do not need certification or may be certified to a lower level [2]. Obtaining a certification only for critical applications can become a difficult task due to the hardware sharing dependency and the diversity of functionalities that the system performs concurrently. The key approach towards a complexity reduction is to prevent the interference between integrated applications both in the temporal and spatial domain [3]. Temporal isolation preserves the timing behavior of applications such that they do not affect one another while executing concurrently on the shared platform [4]. Spatial isolation protects memory elements of applications so they cannot be accessed by the other applications.

For a multi-core platform with shared main memory, spatial isolation between applications can be achieved simply by integrating a Memory Management Unit (MMU) [5] as part of the memory-address translation process. The MMU table is set up to assign a different memory space to each application. In this way, the MMU table protects the given memory space of applications against possible violations from the other ones. Establishing temporal isolation between time-critical applications is a more complex problem. This comes as a consequence of the resource sharing between applications on the same core, as well as from the sharing of resources (main memory and I/O components) between applications running on different cores.

In this paper we present the MultiPARTES[1] platform that provides full temporal isolation for time-critical applications at all levels of a computing system. This platform consists of multi-core hardware with a shared bus on top of which the XtratuM [6] hypervisor executes (Figure 1).

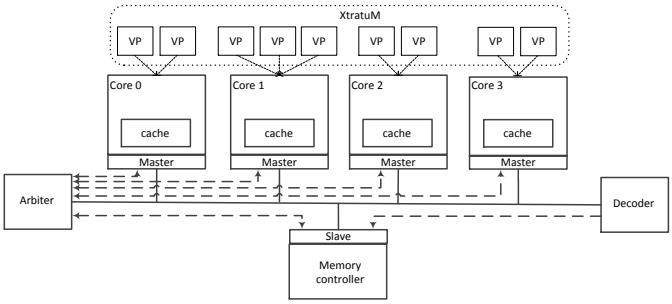

Fig. 1: MultiPARTES Platform

The hypervisor eliminates the possibility of interference between virtual partitions running on the same CPU by implementing a static cyclic scheduling where each virtual partition is associated with a fixed number of CPU cycles. At runtime, each partition is activated based on the static schedule and gets a predefined amount of processor time. Applications of different criticality level are mapped on different virtual partitions. However, the hypervisor is not able to ensure temporal isolation of the critical partitions when they access a shared bus. To eliminate the inter-core interference we have designed a new bus-arbiter scheme that preserves the isolation properties for critical virtual partitions and provides better utilization of shared resources for non-critical ones. The arbiter is based on a hierarchical, two-layer arbitration scheme that switches between a critical and non-critical mode.

The paper is organized as follows. The next section describes the hardware architecture of the MultiPARTES platform, the new bus-arbiter scheme and presents preliminary results on arbiter performance. Section 3 describes the XtratuM hypervisor and its integration to the multi-core platform. Temporal interference and preformance evaluation of the whole platform are shown in Section 4. Section 5 concludes the paper.

II. Hardware Platform for Mixed-Criticality Multicore Systems

The hardware used in MultiPARTES consists of a multi-core LEON3 processor that is interconnected with a memory controller and

[1] www.multipartes.eu

I/O resources through a shared AMBA bus. LEON3 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. Multi-core LEON3 systems are highly customizable with respect to the processor core and periphery features. LEON3 has a seven-stage pipeline with a Harvard architecture, separate caches for instructions and data, a hardware multiplier and divider, on-chip debug support, an MMU with a configurable TLB, and multi-processor extensions [7].

The interconnection between cores with main memory and I/O components is realized through the Advanced Microcontroller Bus Architecture (AMBA). AMBA is an open standard specification that defines an on-chip communication standard for designing high-performance embedded systems [8]. It is widely used in network interconnect chips, RAM controllers, Flash memory controllers and SoCs (System on Chips) [9]. It consists of a high-performance system backbone bus (AHB) on which the CPUs, on-chip memories and other DMA devices reside, and a low bandwidth bus (APB) to which most of the system peripheral devices are connected. The APB is optimized for low power consumption and reduced interface complexity.

The AHB and the APB are connected via a bus bridge. Components which can initiate read or write operations are called AHB master while those which respond to a read/write operation are called AHB slave. Since the bus is shared between all master components, only one master is allowed to initiate a transfer at any time. AMBA uses a centralized arbiter to determine which master gets the right to commence a transfer and also ensures that at any given time at most a single transfer is in progress. The decision for granting the bus is based on an arbitration algorithm. The basic requirements for an arbitration policy are that it should guarantee the fairness of accesses between master components and it has to prevent the starvation of the masters.

A. Dual-Layer Arbitration Scheme

The MultiPARTES platform uses a two-layer arbitration scheme for accesses to the shared-memory bus. The first layer is based on *time division multiple access* (TDMA) and is responsible for guaranteeing the temporal properties of critical partitions, while the other layer employs a *round-robin* (RR) arbitration policy that controls the bus access for non-critical partitions.

TDMA is an arbitration policy that guarantees a fixed bus bandwidth by a priory assigning time slots of fixed length to each master. In contrast, the RR arbitration scheme grants bus access to every master on the bus in a circular manner. A master relinquishes control over the bus when it no longer has data to transfer. When a master has completed its transfer, it passes control to the next master in line [10].

The TDMA scheme is the main bus access driver. Each core that runs time-critical applications has an a-priori assigned time slot to guarantee access on the bus. Individual cores that need higher bandwidth can be assigned multiple slots within the scheduling frame. A few sequential time slots from TDMA arbitration are reserved for cores with non-critical applications. On top of these slots the RR is built. During this RR time interval, dynamic arbitration is activated and all transactions of non-critical cores are competing for the access to the memory bus. To ensure that non-critical cores cannot interfere with the critical ones, an additional rule has to be enforced. The dynamic arbiter (RR) must not serve any request that is issued after the start of the last timeslot in the sequence of the dynamic time slots. Otherwise, such a request could overlap with the next static time slot for the critical cores, thus invalidating the time predictability of the TDMA scheme.

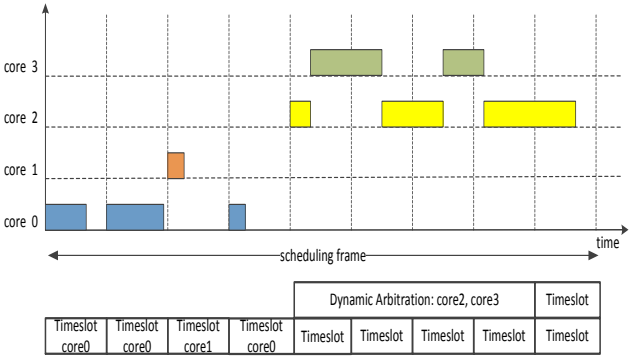

Fig. 2: Dual-policy arbitration scheme

An example of dual-layer arbitration for four-core hardware is presented in Figure 2. The first two cores (core 0 and core 1) are running time-critical applications while the other two cores (core 2 and 3) are executing non-critical ones. The scheduling frame has nine slots where four are assigned to critical cores (core 0 is assigned three slots under the assumption that it needs more bandwidth), the next four slots are reserved for non-critical cores and the last slot is reserved for the completion of ongoing non-critical transactions. Core 0 and 1 are granted to start bus transactions only at the beginning of their assigned slots in order to guarantee the non-interference of transactions. In contrast, cores 2 and 3 are accessing the bus when dynamic arbitration (RR) is active. The last slot, does not allow for any bus access but only serves the completion of the ongoing transaction of core 2.

B. Hardware Architecture of the Dual-Layer Arbiter

The arbiter architecture consists of a TDMA and an RR component (Figure 3). The TDMA components has a wrap incremental counter, a shift register and a controller. The wrap counter is incremented at each clock cycle with a maximal value equal to the number of cycles for one slot. The function of this unit is to shift the token in a shift register at the end of the time slot. The shift register keeps record of the slots. Depending on the active slot, the controller either grants access to an critical request (HBUSREQ0 or HBUSREQ1) or activates/deactivates the RR arbiter.

All request signals from non-critical partitions are connected to the RR-arbiter queue. When the RR arbiter is activated, it grants partitions bus access in the same order as the requests have arrived (HBUSREQ2 or HBUSREQ3).

C. Evaluation of the Arbiter

We implemented and deployed the dual-layer bus arbiter on an FPGA development kit (terasic DE2-115) to demonstrate the feasibility of the Dual-Layer arbiter and to evaluate its performance. The arbiter is written in VHDL language and deployed as part of the

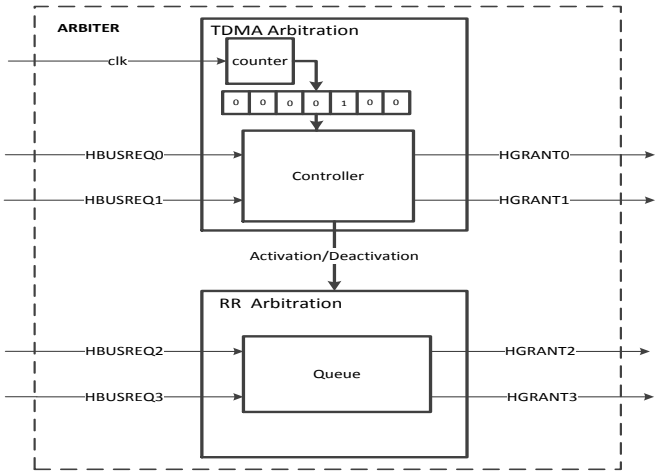

Fig. 3: Arbiter architecture for mixed-criticality systems

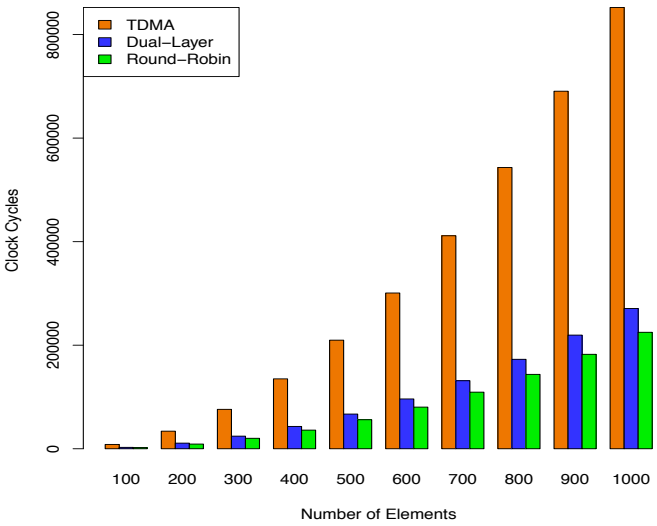

Fig. 4: Execution time of bubble sort algorithm

Grlib IP library [11]. The interconnection with the memory controller and the I/O resources is done through the AMBA bus. The bus is configured not to support split transactions.

For this evaluation a bubble sort algorithm was chosen. The algorithm sorts vectors of sizes from 100 to 1000 elements. In this evaluation, the algorithm is executed on a bare-metal processor, without any operating system in order to avoid the overhead generated from the hypervisor. All four cores were executing the same code. The goal was to test the temporal isolation for time-critical cores and also to compare the execution performance of the noncritical partitions. For comparison we used three different bus arbitration policies: pure TDMA, RR and our Dual-Layer (DL) policy.

The TDMA scheduling frame consisted of four slots and each slot was assigned to a core. The DL arbiter used five slots where the critical CPUs (CPU0 and CPU1) were assigned to the first two slots (slot 0 and 1) while the non-critical CPUs were allowed on the following two slots. No arbitration was allowed on the fifth slot (see above). The slot size was 20 clock cycles (the longest transfer was taking 18 cycles).

Figure 4 shows the observed execution time of the bubble sort algorithm for different vector sizes. The execution time of non-critical partitions is illustrated for TDMA, RR and DL. From the results we see that the execution time of the algorithm in non-critical cores with DL arbitration is 3.1 time shorter than in a system with pure TDMA. Compared to pure RR, the execution of bubble sort with DL lasts 1.2 times longer. For critical partitions the execution time with TDMA and DL is the same.

In summary, the results of this experiment suggest that DL arbitration improves the performance of non-critical partitions without affecting the time properties of the critical ones.

III. The XtratuM Hypervisor

A. Introduction of XtratuM

XtratuM [12], [13] is a bare-metal hypervisor specifically designed for embedded real-time systems that uses para-virtualization techniques to emulate hardware behaviour. The para-virtualized model offers potential performance benefits when a guest operating system or application is aware that it is running within a virtualized environment, and it has been modified to exploit this. One potential downside of this approach is that such modified guests cannot ever be migrated back to run on physical hardware.

XtratuM has been designed to achieve real-time constraints with a set of properties that strongly follow certification issues. These properties can be summarised as:

- Spatial isolation: A partition is completely allocated to isolated memory regions. The hypervisor guarantees the spatial isolation of the partitions.
- Temporal isolation: A partition is executed at specified and fixed temporal intervals. A cyclic scheduling policy is implemented by the hypervisor. The temporal allocation of time to a partition is not impacted by the execution of other partitions, although shared resources could produce interference in the execution time duration of its activities.
- Predictability: A partition with real-time constraints has to execute its code in a predictable way. It can be influenced by the underlying layers of software (guest-OS and hypervisor) and by the hardware. From the hypervisor point of view, the predictability applies to the provided services, the operations involved in the partition execution and internal operations (partition context switch, interrupt management, etc.).
- Fault isolation and management: Fault management is a fundamental aspect in critical systems, and it is strongly related with certification issues. Faults, when they occur, are detected and handled via Health Monitor which is statically configured.
- Static resource allocation: The system architect is responsible for the system definition and resource allocation. This

system definition is detailed in the system's configuration file, which specifies all system resources, namely number of CPUs, memory layout, peripherals, partitions, the execution plan of each CPU, etc.

B. Integration on the Multicore Platform

XtratuM has been adapted to multi-core systems [14] based on LEON4 processors and x86 and LEON3-bicore in the MultiPARTES project [15], [16]. In the multi-core approach, the hypervisor can provide several virtual CPUs to the partitions. A partition can be mono or multi-core. Different partitions (from the point of view of the number of cores) can coexist in the system. This approach allows profiting from a multi-core platform, even if the partitions are not multi-core by building multi-core or monocore partitions.

In order to handle the underlying multi-core hardware it can be configured following an Asymmetric Multiprocessing (AMP) or Symmetric Multi-Processor (SMP). In AMP, there is an instance of the hypervisor running on each core, which executes the allocated partitions. In the SMP approach, one single hypervisor instance manages all hardware resources. While the AMP software architecture simplifies the hypervisor, the SMP approach permits the use of mono or multi-core execution environments, and offers higher flexibility to assign partitions to different cores. Moreover, mono-core partitions in SMP architectures can be permanently allocated to one core, or several on different partition activations with no temporal overlap.

In the adaptation of XtratuM to multi-core platforms, the hypervisor model has been re-designed to support the concept of virtual CPU (vCPU). Virtual CPUs are abstractions that model hardware CPU behaviour and are managed in an analogous way, but can be allocated to any of the existing cores. There are as many virtual CPUs on the system as physical cores and they behave on a similar manner: when a partition starts its execution, only one vCPU is active, being responsibility of the partition to initialize the remaining vCPUs. To this end, it has been necessary to extend XtratuM with new hypercalls that allow the partitions to manage virtual CPUs operation.

In a multi-core partitioned system partitions can use one core (mono-core partitions) or several cores (multi-core partitions). Several mapping schemes can be considered:

- Each monocore partition is mapped to one core
- Several monocore partitions are mapped to one core
- A mono-core partition is mapped to different cores at different time intervals
- A multi-core partition is mapped to several cores at the same temporal intervals

Figure 5 shows a configuration with a 3 mono-core and 1 multi-core partitions. Partitions P1, P2 and P3 allocates their virtual core (vCPU0) to one of the real cores (CPU0 or CPU1). P3 uses both cores at different intervals. P4 is a multi-core partition (two virtual cores) and requires the use of both real cores.

C. Hypervisor scheduling

XtratuM can associate a scheduling policy to each core or group of cores. Two scheduling policies are implemented: cyclic scheduling and priority based. The policy is statically specified in the configuration file. In the cyclic scheduling, the configuration file

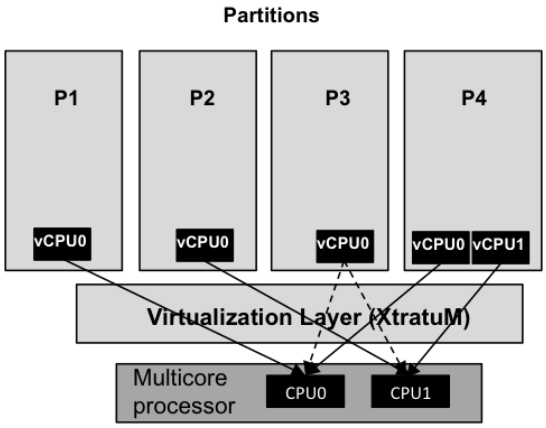

Fig. 5: Partitioned architecture in multi-core platforms.

specifies the temporal windows or slots in a major frame (MAF) where partitions will be scheduled. Each slot details the partition, the execution interval (as offset from the start of the major time frame and the duration) and the virtual CPU. Partitions definition includes the number of virtual CPUs to be used.

In case of priority based scheduling, partitions, allocated to the core that uses this policy, have to specify the priority, period and budget.

Next listing shows the specification of a cyclic schedule of four partitions in two cores according to the partition mapping shown in Fig 5.

XML configuration file: schedule specification

```
<ProcessorTable>
 <Processor id="0">
  <CyclicPlanTable>
   <Plan id="0" majorFrame="20ms">
    <Slot id="0" start ="0ms" duration="3ms" partitionId ="1" vCpuId="0"/>
    <Slot id="1" start ="3ms" duration="3ms" partitionId ="3" vCpuId="0"/>
    <Slot id="2" start ="7ms" duration="3ms" partitionId ="1" vCpuId="0"/>
    <Slot id="3" start ="12ms" duration="3ms" partitionId ="1" vCpuId="0"/>
    <Slot id="4" start ="15ms" duration="5ms" partitionId ="4" vCpuId="0"/>
   </Plan>
  </CyclicPlanTable>
 </Processor>
 <Processor id="1">
  <CyclicPlanTable>
   <Plan id="0" majorFrame="20ms">
    <Slot id="0" start ="0ms" duration="6ms" partitionId ="2" vCpuId="0"/>
    <Slot id="1" start ="7ms" duration="3ms" partitionId ="3" vCpuId="0"/>
    <Slot id="2" start ="10ms" duration="3ms" partitionId ="2" vCpuId="0"/>
    <Slot id="3" start ="13ms" duration="2ms" partitionId ="3" vCpuId="0"/>
    <Slot id="4" start ="15ms" duration="5ms" partitionId ="4" vCpuId="1"/>
   </Plan>
  </CyclicPlanTable>
 </Processor>
</ProcessorTable>
```

Figure 6 draws the execution chronogram of this example. P1 and P2 allocate their vCPU0 to CPU0 and CPU1, respectively. P3 allocates its vCPU0 to CPU0 and CPU1 at different time intervals (no overlap). P4 maps its virtual cores to the real cores at the same time intervals.

The configuration file is statically defined off-line. A set of techniques and tools are required to generate the schedule, according to the partition temporal requirements, criticality level, platform needs, etc., and to verify the coherence and correctness of the final schedule.

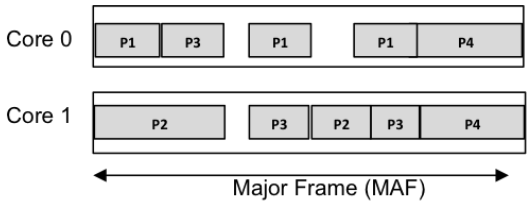

Fig. 6: Scheduling scheme.

In [17], a configuration and scheduling tool is presented. The hypervisor requires a binary representation of a verified configuration file to execute the system.

The main challenge in multi-core hypervisor is to deal with the temporal inteference when several cores are executing code at the same time. From the scheduling point of view, the temporal isolation in multi-core can consider two aspects:

- temporal allocation of systems resources: partition execution is statically defined (temporal windows).
- temporal interference: impact of the shared resources use by other cores.

While the hypervisor can guarantee the temporal allocation of resources, it requires the hardware support to deal with the temporal interference.

IV. Temporal Interference and Performance Analysis

In this section, the performance of the hypervisor layer comparing both architectures proposed in the MultiPARTES project for the AMBA bus: RR and TDMA is evaluated. The target is a LEON3 dual-core at 50MHz with DDR memory.

A. Temporal Interference Analysis

Temporal interference is produced when partitions in different cores use shared resources. We focus on this evaluation on 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 partitions running in diffeent cores is defined. The scenario is defined with two 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 performs a loop that read and modify the contents of a table. 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 both partitions to access physically to memory.

This scenario is executed under the following scheduling plan:

- MAF: 300 msec
- Payload cost 20 msec (approx).
- P_1 slot duration: 150 msec.
- P_2 slot duration: 150 msec.
- Experiments:
  - S0: No interference.
  - S25: 25% of interference.
  - S50: 50 % of interference.
  - S75: 75 % of interference.
  - S100: 100 % of interference.

Fig. 7 shows the schedule of S25.

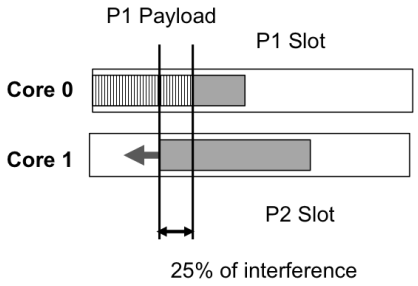

Fig. 7: Schedule of the S25

Table I shows the results of the impact of P_2 on the execution of P_1 in the LEON3 bi-core platform with RR bus arbitration. Under an approximated overlapping of the 25%, the increment of the execution time is in average 2899 μsecs (23544 - 20645). Next table presents the statistics of 100 executions of each scenario.

TABLE I: Impact of the memory accesses in the $P1$ execution time RR bus arbitration

Exec. Time (μsecs)	S0	S25	S50	S75	S100
Avg	20645	23544	27181	30577	33418
Max	20700	23557	27326	30614	33691
Min	20698	23305	27253	30535	33596
Stdev	0.63	12.46	16.77	17.19	22.81
Inteference	0%	14%	32%	48%	62%

Results show the impact of the interference when partitions allocated to different cores are executed with a level of overlapping. If both partitions are executed at the same time, the interference can produce an increment of 62% in the execution of P1.

Same experiment has been executed in the LEON3 bi-core platform with TDMA bus arbitration. Table II shows the results.

TABLE II: Impact of the memory accesses in the $P1$ execution time TDMA bus arbitration

Exec. Time (μsecs)	S0	S25	S50	S75	S100
Avg	120099	120099	120099	120099	120099
Max	120100	120100	120099	120100	120100
Max	120097	120096	120098	120098	120098
Stdev	0.99	1.03	0.97	0.96	0.96
hline Inteference	0%	0%	0%	0%	0%

These results show that the impact, as expected, depends on the overlapping interval and the bus arbitration policy. While the

temporal interference has a very high impact in the first hardware platform, the new design based on the TDMA arbitration policy fully achieves the temporal isolation of partitions. This is a relevant result for temporal and spatial partitioning platforms that permits to execute independently applications in multi-core systems.

B. Performance Analysis

In this section, a comparison of the performance of two hardware solutions is performed. Performance analysis includes two parameters: temporal cost of the same activity and hypervisor partition context switch in both platforms.

Table III summarises the temporal costs of the same payload executed in previous experiments.

TABLE III: Temporal cost comparison

	Avg Time (μsecs)
TDMA bus arbitration	120099
RR bus arbitration	20689
Increment cost factor	5.80

The use of a TDMA based arbitration policy introduces delays in the partition execution. For the configuration experimented in this paper, the computation time of a payload is incremented 5.8 times which can be relevant depending on the timing requirements of real-time tasks.

On the other hand, the partition context switch of the hypervisor measures the time required by the hypervisor to switch from one partition to another in a core. This cost has been experimentally measured by instrumenting the hypervisor code to annotate the entry and exit to the partition context switch service. Table IV shows the results in μsecss for both platforms.

TABLE IV: PCS comparison

	Avg Time (μsecs)
TDMA bus arbitration	1042
RR bus arbitration	224
Increment cost factor	4.65

The observed increment of the PCS is 4.65 times. While 224 μsecss for the partition context switch in space applications with period ranges in the order to dozens of milliseconds can be acceptable, the cost of 1 millisecond for the PCS introduces high overheads and reduces the period ranges to hundreds of milliseconds.

V. Conclusion

In mixed-criticality systems, critical applications are subject of certification. Without proper isolation of time-critical applications the process for certifying can become complex and time consuming.

In this paper we describe a memory architecture that provides temporal isolation between virtual partitions. Implementing the dual-layer memory-bus arbiter helps the hypervisor to guarantee time bounds for critical applications and to improve the performance of non-critical applications when they access the shared memory bus. We also demonstrate the feasibility of the proposed memory hierarchy by implementing it in an FPGA and running XtratuM on top of that hardware. The evaluation proves that even when critical partitions execute with full temporal overlap, the temporal properties of each of these partitions are preserved.

In future work we will run parallel applications on the multi-core hardware in order to evaluate the utilization of the hardware at the entire multi-core system level and experiment with the dual-layer arbiter.

Acknowledgment

This research was partially funded under the European Union's 7th Framework Programme under grant agreement no. 287702: Multicores Partitioning for Trusted Embedded Systems (MultiPARTES).

References

[1] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, 2010, pp. 13–22.

[2] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011, pp. 34–43.

[3] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic, "Idamc: A many-core platform with run-time monitoring for mixed-criticality," in *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*. IEEE, 2012, pp. 24–31.

[4] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "Flexpret: A processor platform for mixed-criticality systems," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS). IEEE*, 2014.

[5] D. A. Patterson and J. L. Hennessy, "Computer organization and design," *Morgan Kaufmann*, 2007.

[6] M. Masmano, I. Ripoll, A. Crespo, and J.-J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *11th Real-Time Linux Workshop*, 2009.

[7] A. Gaisler and S. Göteborg, "Leon3 multiprocessing cpu core," *Aeroflex Gaisler, February*, 2010.

[8] A. A. Specification, "Multi layer ahb specification,(rev2. 0)," 2001.

[9] Y. Godhal, K. Chatterjee, and T. A. Henzinger, "Synthesis of amba ahb from formal specification: a case study," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5-6, pp. 585–601, 2013.

[10] S. Pasricha and N. Dutt, *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[11] Leon3. [Online]. Available: http://www.gaisler.com/

[12] M. Masmano, I. Ripoll, S. Peiró, and A. Crespo, "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), 19-21 May 2010.

[13] M. Masmano, I. Ripoll, A. Crespo, and J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *Eleventh Real-Time Linux Workshop*, Dresden (Germany), 28-30 September 2009.

[14] E. Carrascosa, J. Coronel, M. Masmano, P. Balbastre, and A. Crespo, "Xtratum hypervisor redesign for LEON4 multicore processor," *SIGBED Review*, vol. 11, no. 2, pp. 27–31, 2014. [Online]. Available: http://doi.acm.org/10.1145/2668138.2668142

[15] "Multipartes: Multi-cores partitioning for trusted embedded systems," 2011. FP7 ICT 287702 European Project. (http://www.multipartes.eu).

[16] S. Trujillo, A. Crespo, and A. Alonso, "Multipartes: Multicore virtualization for mixed-criticality systems," in *2013 Euromicro Conference on Digital System Design, DSD 2013, Los Alamitos, CA, USA, September 4-6, 2013*, 2013, pp. 260–265.

[17] V. Brocal, M. Masmano, I. Ripoll, A. Crespo, and P. Balbastre, "Xoncrete: a scheduling tool for partitioned real-time systems," in *Embedded Real-Time Software and Systems*, 2010.