# SPACE AND TIME PARTITIONING WITH HARDWARE SUPPORT FOR SPACE APPLICATIONS

**S. Pinto**[1], **A. Tavares**[1], **and S. Montenegro**[2]

[1]*Centro Algoritmi, Universidade do Minho, Guimaraes, Portugal*
[2]*Aerospace Information and Technology, Universität Würzburg, Germany*

## ABSTRACT

Complex and critical systems like airplanes and spacecraft implement a very fast growing amount of functions. Typically, those systems were implemented with fully federated architectures, but the number and complexity of desired functions of todays systems led aerospace industry to follow another strategy. Integrated Modular Avionics (IMA) arose as an attractive approach for consolidation, by combining several applications into one single generic computing resource. Current approach goes towards higher integration provided by space and time partitioning (STP) of system virtualization. The problem is existent virtualization solutions are not ready to fully provide what the future of aerospace are demanding: performance, flexibility, safety, security while simultaneously containing Size, Weight, Power and Cost (SWaP-C).

This work describes a real time hypervisor for space applications assisted by commercial off-the-shell (COTS) hardware. ARM TrustZone technology is exploited to implement a secure virtualization solution with low overhead and low memory footprint. This is demonstrated by running multiple guest partitions of RODOS operating system on a Xilinx Zynq platform.

Key words: Space and Time Partition; Virtualization; TrustZone; Real-Time; Embedded Systems; ARM.

## 1. INTRODUCTION

The market of complex and critical systems like cars, airplanes and spacecraft have experienced unprecedented growth over the last few years and is expected to continue growing exponentially for the foreseeable future [1]. The number and complexity of desired functions evolved in such a way that fully federated architectures, where each function is implemented in its own embedded controller, become completely impracticable. Naturally, industries rapidly tried to find other alternatives, and aeronautics pioneering the shift from traditional federated architectures to an IMA [2] architecture. By combining several applications into one generic powerful computing resource, they were able to get a reduction on SWaP-C.

As space domain typically shares the same basic needs of aeronautics, they rapidly concluded that IMA strategy could be spun-in to the space domain. The problem was that the use of generic platforms altogether with several COTS-based components with different criticality and from several suppliers, imposed integration challenges namely in terms of reusability and safety. The introduction of STP [3, 4] for separation of concerns between functionally independent software components was the solution to achieve higher level of integration, while maintaining the robustness of federated architectures. By containing and/or isolating faults, STP approach eliminates the need to re-validate unmodified applications on an IMA system, because the guaranteed isolation it provides limits re-certification efforts only at the partition level.

Virtualization technology has been used as an implementation technique to provide STP. Over the last few years several works have been proposed in the aerospace domain [5, 6, 7, 8, 9, 10, 11, 12]. Some of them following a (software-based) full-virtualization approach, while others implementing para-virtualization. Between both approaches there is a trade-off between flexibility and performance: full-virtualization incurs in a higher performance cost, while para-virtualization incurs in a higher design cost. Apart from performance and flexibility, safety and, more recently, security became important requirements that are driving the development of current and next generation of safety-critical systems. Research and industry focused so their attention on providing hardware support to assist and secure virtualization from the outset. Intel, ARM and Imagination/MIPS introduced their own COTS technologies, and naturally several hardware-based solutions have been proposed [13, 14, 15, 16, 17], although none of them specifically designed for the aerospace domain. Among existent COTS technologies, ARM TrustZone [18] is gaining particular attention due to the ubiquitous presence of TrustZone-enabled processors in the embedded sector. The problem is existent TrustZone-based solutions [13, 14, 16] not only fail in providing ability for running an arbitrary number of partitions (they mainly rely on a dual-OS configuration), but also they (i.e., TrustZone-based solutions) were not designed taking into consideration the real time requirements of safety-critical systems.

Our work goes beyond state-of-art presenting a secure virtualization solution assisted by COTS hardware. A distinctive aspect of our hypervisor is the use of ARM TrustZone technology to assist and secure real-time virtualization for space. This is demonstrated by running several unmodified RODOS OS partitions on a hybrid Xilinx ZC702 platform with really low overhead and low memory footprint. Furthermore, with the recent announcement of ARM about their decision to introduce TrustZone technology in all Cortex-M and Cortex-R processors series, we strongly believe this solution as more than an isolated implementation for a specific platform. From our point of view, it will also provide the foundation to drive next generation virtualization solutions for middle and low-end space applications (e.g., small satellites).

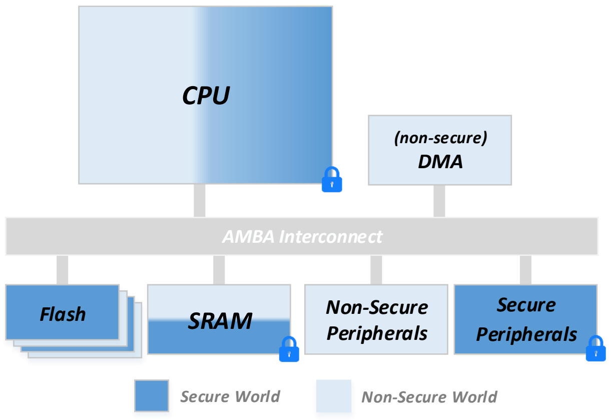

*Figure 1. TrustZone hardware architecture*

## 2. RELATED WORK

The idea beyond implementation of STP through virtualization in the aerospace domain is not new, and the first attempts to start exploiting this technology were proposed some years ago.

XtratuM [5] [6] was one of the first well known hypervisors implemented to meet the safety critical requirements of aerospace domain. Implemented in its original form [5] as a Loadable Kernel Module (LKM) for x86 architectures, it naturally evolved to be independent and bootable. The result was XtratuM 2.0 [6], a type 1 (bare-metal) hypervisor which employes para-virtualization techniques and a strong effort in redesign to be closer to the ARINC-653 standard. XtratuM was ported to reference platforms for the spatial sector like LEON2 and LEON3, and recently it was extended with multicore support under the MultiPARTES project [7].

Steven VanderLeest start by developing an early prototype of an ARINC 653 implementation using the virtualization technology of the open source Xen hypervisor along with a Linux-based domain/partition OS [8]. The hypervisor, called as ARLX (ARINC 653 Real-time Linux on Xen) [9], was later extended with more safety and security capabilities, basically by applying formal analysis techniques as well as certification. A distinctive aspect of ARLX was its innovative approach to multicore processors, by scheduling the I/O system partition in one core and all the guest partitions on remaining cores. More recently, in [19] Steven VanderLeest (with cooperation of Big Players like Xilinx, Mentor Graphics and Lynx Software) introduced some ideas of how to address the future of avionics. He basically proposes the use of what we almost have already implemented: secure virtualization deployed on heterogenous multicore platforms (HMP). He conducted his discussion around the Xilinx Zynq Ultrascale+.

Hyunwoo Joe et al. [10] presented a prototype of a type 2 (hosted) full-virtualized hypervisor with kernel-level ARINC 653 partitioning. The hypervisor was implemented as an extension to the embedded Linux kernel with some parts located into the kernel while others implemented as system processes. Several optimization techniques were used to alleviate the trap-and-emulate overhead, however despite those efforts the prototype still showed a considerable performance degradation. Authors promoted reusability in detriment of performance, in order to not modify certified flight software from previous missions. The hypervisor was the first known implementation targeting dual-core LEON4-based platforms.

RodosVisor [11, 12], developed by Tavares et al., is a bare-metal hypervisor supporting both full and para-virtualization, alongside with real-time capabilities. The hypervisor is able to execute several applications according to the IMA model, running on a Xilinx FPGA board with built-in PowerPC core. It is an ARINC 653 quasi-compliant hypervisor, since it implements ARINC 653 services but, not strictly following the APIs standard. A distinctive aspect of RodosVisor is its object-oriented approach and the possibility of customization using generative programming techniques.

There are also other relevant academic and commercial virtualization solutions, and Gu and Zhao [20] survey a number of them for a variety of embedded domains (not just for aerospace). They basically review hypervisors based on Xen, the Kernel-based Virtual Machine (KVM) and L4. Several works trying to make use of COTS-hardware to assist virtualization have been proposed [14, 15, 16, 17], but none of them was specifically designed for the aerospace domain.

## 3. ARM TRUSTZONE

TrustZone technology refers to security extensions available in all ARM Application-processors (Cortex-A) since ARMv6 architecture. These hardware security extensions virtualize a physical core as two virtual cores, providing two completely separated execution environments:

the *secure* and the *non-secure* worlds (Fig. 1). An extra 33rd bit - the NS (Non-Secure) bit - indicates in which world the processor is currently executing. To switch between the secure and the non-secure world, a special new secure processor mode, called *monitor mode*, was introduced. To enter in the monitor mode, a new privileged instruction was also specified - SMC (*Secure Monitor Call*). The monitor mode can also be enabled by configuring it to handle interrupts and exceptions in the secure side. The memory infrastructure can be also partitioned into distinct memory regions, each of which can be configured to be used in either worlds or both. The processor provides also two virtual Memory Management Units (MMUs), and isolation is still available at the cache-level. The AXI (Advanced eXtensible Interface) system bus carries extra control signals to restrict access on the main system bus, which enables TrustZone architecture to secure also peripherals. The Generic Interrupt Controller (GIC) supports both secure and non-secure prioritized interrupt sources.

## 4. HYPERVISOR

RTZVisor (Real Time TrustZone-assisted Hypervisor) is a bare-metal hypervisor carefully designed to meet the specific requirements of real time space applications. Exploiting COTS ARM TrustZone technology, it is possible to implement strong spatial and temporal isolation between guests. All data structures and hardware resources are pre-defined and configured at design time, and devices and interrupts can be directly managed by specific guest partitions. Exceptions and errors are managed through a special component called Health Monitor, which is able to recover guests from undefined states. Fig. 2 depicts the complete system architecture: RTZVisor runs in the most privileged mode of the secure world side, i.e., monitor mode, and has the highest privilege of execution; unmodified guest OSes can be encapsulated between the secure and non-secure world side - the active guest partition runs in the non-secure world side, while inactive guest partitions are preserved in the secure world side; for active guest the RTOS runs in the kernel mode, while RT applications run in user mode.

### 4.1. CPU

TrustZone technology virtualizes each physical CPU into two virtual CPUs: one for the secure world and other for the non-secure world. Between both worlds there are an extensive list of banked registers. Typically, existent TrustZone-based solutions implement only dual-OS support, where each guest is running in a different world. In this particular case, the virtual CPU support is guaranteed by the hardware itself and therefore each world has its own virtual hard-processor.

Our system is completely different. Since it is able to support an arbitrary number of guest partitions, all of them executed from the non-secure side (once at a time), dictating the sharing of the same virtual hard-processor, supported by software. For that reason, the virtual soft-processor state (vCPU) of each guest should be preserved. This virtual soft-processor state includes the core registers for all processor modes (vCore), the CP15 registers (vCP15) and some registers of the GIC (vGIC), encompassing a total of 55 registers. RTZVisor offers as many vCPUs as the hardware provides, but only a one-to-one mapping between vCPU, guest and real CPU is supported.

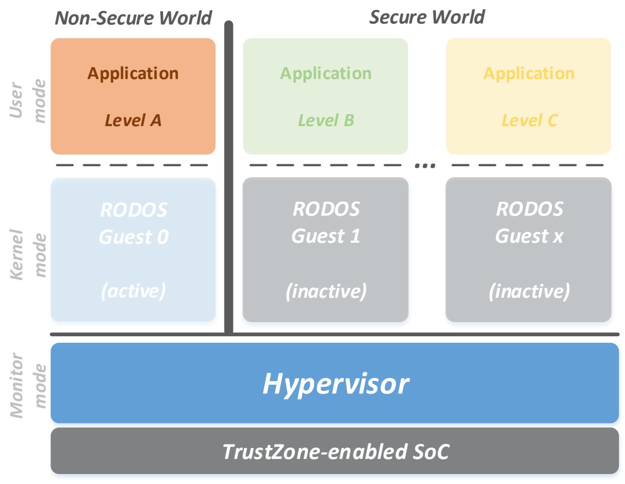

*Figure 2. System Architecture*

### 4.2. Memory

Traditional hardware-assisted memory virtualization relies on Memory Management Unit (MMU) support for 2-level address translation, mapping guest virtual to guest physical addresses and then guest physical to host physical addresses. This MMU feature is a key feature to run unmodified guest OSes, and also to implement isolation between them.

TrustZone-enabled system on chips (SoCs) only has MMU support for single-level address translation. Nevertheless, they provide a component called TrustZone Address-Space Controller (TZASC) which allows partition of memory into different segments. This memory segmentation feature can be exploited to guarantee strong spatial isolation between guests, basically by dynamically changing the security state of their memory segments. Only the guest partition currently running in the non-secure side should have its own memory segment configured as non-secure, and the remaining memory as secure. If the running guest tries to access a secure memory region (belonging to an inactive guest partition or either the hypervisor), an exception is automatically triggered and redirected to the hypervisor. Since only one guest can run at a time, there is no possibility of the inactive guests (belonging momentously to the secure side) to change the state of another guest.

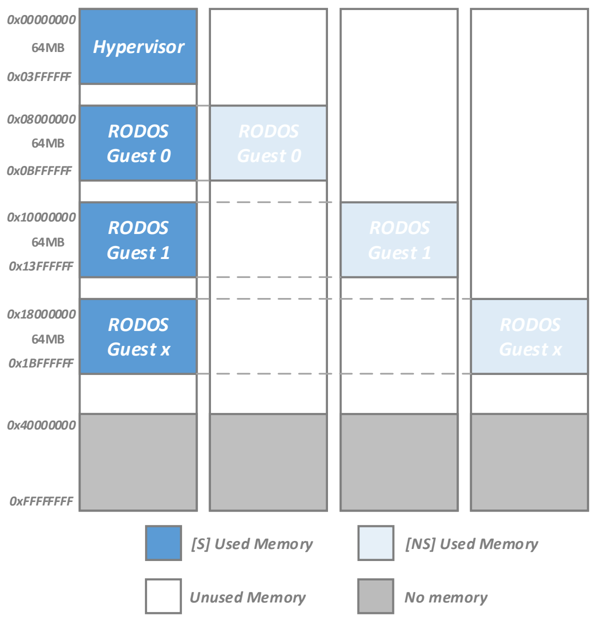

*Figure 3. System memory*

Memory segments can be configured with a specific granularity, which is implementation defined, depending on the vendor. In the hardware under which our system was deployed, Xilinx ZC702, memory regions can be configured with a granularity of 64MB, which mean for a memory of 1GB it is possible to isolate a total of 15 guest partitions (one memory segment is for the hypervisor itself). Our system relies on the TZASC to implement isolation between guests, and MMU supporting only single-level address translation. It means that guests have to know the physical memory segment they can use in the system, requiring relocation and consequent recompilation of the guest OS. Fig. 3 depicts the memory setup and respective secure/non-secure mappings, for a virtualized system consisting in the hypervisor and three guest partitions. In this specific configuration, the hypervisor uses the first memory segment (0x00000000 - 0x03FFFFFF), and has access to all memory. RODOS Guest-0 uses the third 64MB memory segment, and is only allowed to access one non-secure memory segment (0x08000000 - 0x0BFFFFFF); RODOS Guest-1/x are mapped the same way, but within their respective memory segment.

### 4.3. Scheduler

RTZVisor implements a cyclic scheduling policy, to ensure one guest partition cannot use the processor for longer than its defined CPU quantum. The time of each slot can be different for each guest, depending on its criticality classification, and is configured at design time. By adopting a variable time slot strategy instead of a multiple fixed approach, the hypervisor interference is minimized and it is ensured higher performance and deterministic execution, because guest is only interrupted when the complete slot is over.

### 4.4. Devices

TrustZone technology allows devices to be (statically or dynamically) configured as secure or non-secure. This hardware feature allows the partition of devices between both worlds while enforcing isolation at the device level.

RTZVisor implements device virtualization adopting a pass-through policy, which means devices are managed directly by guest partitions. To ensure strong isolation between them, devices are not shared between guests and are assigned to respective partitions at design time. To achieve this strong isolation at device level, devices assigned to guest partitions are dynamically configured as non-secure or secure, depending on its state (active or inactive). This guarantees an active guest cannot compromise the state of a device belonging to another guest, and if an active guest partition tries to access a secure device then an exception will be automatically triggered and handled by RTZVisor. Devices assigned to the hypervisor itself (e.g., Hypervisor timer) are always configured as secure and can never be directly accessed by any guest.

### 4.5. Interrupts

In TrustZone-enabled SoCs, the GIC supports the coexistence of secure and non-secure interrupt sources. It allows also the configuration of secure interrupts with a higher priority than the non-secure interrupts, and has several configuration models that allow to assign IRQs and FIQs to secure or non-secure interrupt sources.

RTZVisor configure interrupts of secure devices as FIQs, and interrupts of non-secure devices as IRQs. Secure interrupts are redirected to the hypervisor, while non-secure interrupts are redirected to the active guest (without hypervisor interference). When a guest partition is under execution, only the interrupts managed by this guest are enabled, which minimizes inter-partition interferences through hardware. Interrupts of inactive guest partitions are momentaneously configured as secure, but disabled. This mean the interrupt is pending in the distributor, but is not forwarded to the interface. As soon as the respective guest become active, the interrupt will be then processed. The prioritization of secure interrupts avoid active guest partition to perform a denial-of-service attack against the secure side (hypervisor).

### 4.6. Time

Temporal isolation in virtualized systems is typically achieved using two levels of timing: the hypervisor level and the guest level. For the guest level, hypervisors typically provide timing services which allow guests to have notion of virtual or real time. For mission critical real time systems, where time-responsiveness plays a critical role, guest partitions have necessarily to keep track of the wall-clock time.

*Table 1. Health monitoring events and actions*

| ***Event name*** | ***Hypervisor*** *pre-def. action* | ***Guest*** *pre-def. action* |
|---|---|---|
| **Guest triggered** | | |
| *DATA_ABORT* | — | Reboot |
| *PREF_ABORT* | — | Reboot |
| *UNDEF_INST* | — | Reboot |
| **Hypervisor triggered** | | |
| *MEM_VIOL* | Log | Reboot |
| *DEV_VIOL* | Log | Reboot |
| *NO_GUESTS* | Reset | — |

RTZVisor implements also two levels of timing: it has internal clocks for managing the hypervisor time, and internal clocks for managing the guest partitions time. The timers dedicated to the hypervisor are configured as secure devices, i.e., they have higher privilege of execution than the timers dedicated to the active guest. This means that despite of what is happening in the active guest, if an interrupt of a timer belonging to the hypervisor is triggered, the hypervisor takes control of the system. Whenever the active guest is executing, the timers belonging to the guest are directly managed and updated by the guest on each interrupt. The problem is how to deal and handle time of inactive guests. For inactive guests the hypervisor implements a virtual tickless timekeeping mechanism based on a time-base unit that measures the passage of time. Therefore, when a guest is rescheduled, its internal clocks and related data structures are updated with the time elapsed since its previous execution.

### 4.7. Health Monitor

The Health Monitor (HM) component is the module responsible for detecting and reacting to anomalous events and faults. Although at an early stage of development, once an error or fault is detected, RTZVisor reacts to the error providing a simple set of predefined actions. The complete list of events and pre-defined actions can be seen in Tab. 1. For example, at the hypervisor level, if a guest tries to access a portion of memory outside its boundaries, the hypervisor detects and registers the space violation and immediately reboots the guest.

## 5. EVALUATION

Our solution was evaluated on a Xilinx ZC702 board targeting a dual ARM Cortex-A9 running at 600MHz. In spite of using a multicore hardware architecture, the current implementation only supports a single-core configuration. During our evaluation, MMU, data and instruction cache and branch prediction (BP) support for guests were disabled. We focused on the following metrics: (i) memory footprint, (ii) context-switch time and (iii) performance.

*Table 2. Memory footprint results (bytes)*

| | ***.text*** | ***.data*** | ***.bss*** | ***Total*** |
|---|---|---|---|---|
| ***Hypervisor*** | 5568 | 192 | 0 | **5760** |

*Table 3. Context-switch evaluation (microseconds)*

| ***Context-switch operation*** | ***Time ($\mu s$)*** |
|---|---|
| *Timer interrupt management* | *1.620* |
| *Save vCore context* | *1.873* |
| *Scheduler* | 4.000 |
| *vGIC context-switch* | 31.533 |
| *Time management* | 53.033 |
| *Memory configuration* | 1.053 |
| *Restore vCore context* | 1.963 |
| ***TOTAL*** | **95.075** |

### 5.1. Memory footprint

To access memory footprint results we used the size tool of ARM Xilinx Toolchain. Tab. 2 presents the collected measurements, where boot code and drivers were not take into consideration. As it can be seen, the memory overhead introduced by the hypervisor - and in fact the trusted computing base (TCB) of the system - is really small, i.e., around 6KB. The main reasons behind this low memory footprint are (i) the hardware support of TrustZone technology for virtualization and (ii) the careful design and static configuration of each hypervisor component.

### 5.2. Context switch

To evaluate the guest context switch time we used the Performance Monitor Unit (PMU) component. To measure the time consumed by each internal activity of the context-switch operation, breakpoints were added at the beginning and end of each code portion to be measured. Results were gathered in clock cycles and converted to microseconds accordingly to the processor frequency (600MHz). Each value represents an average of ten collected samples.

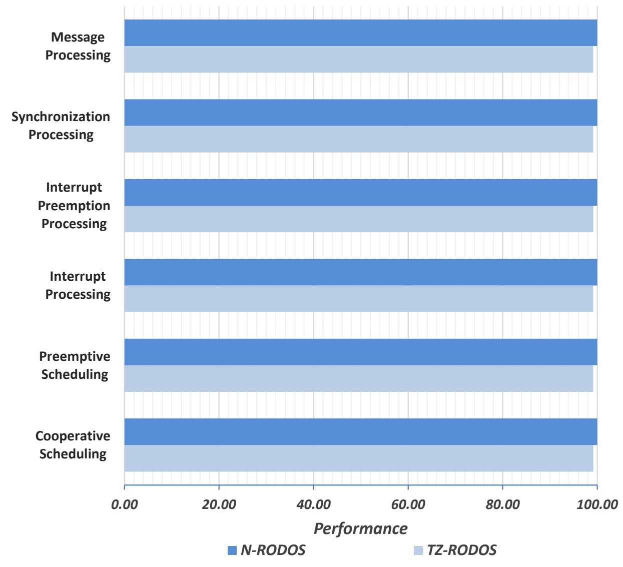

*Figure 4. Thread-Metric benchmarks results*

The list of activities as well as the measured time are presented in Tab. 3. As it can be seen, the activities which present higher consuming time are the virtual GIC context-switch and the time management. In both cases, there is a chance to optimize those operations, because our current solution is more focused on generalization instead of particularization. This mean that context-switch operation was implemented to be as much OS-independent as possible, and such a design decision directly and negatively impact interrupt and time management. We are completely sure that if we redesign those parts to just fit with RODOS, the impact on performance will be significantly smaller.

## 5.3. Performance

The Thread-Metric Benchmark Suite consists of a set of benchmarks specific to evaluate RTOSes performance. The suite comprises 7 benchmarks, evaluating the most common RTOS services and interrupt processing: cooperative scheduling; preemptive scheduling; interrupt processing; interrupt preemption processing; synchronization processing; message processing; and memory allocation. For each benchmark the score represents the RTOS impact on the running application, where higher scores correspond to a smaller impact.

For the first part of the experiment, RTZVisor was configured with a 10 milliseconds guest-switching rate. The system was set to run one single guest partition, and the hypervisor scheduler was forced to reschedule the same guest, so that results can translate the full overhead of the complete guest-switching operation. We ran benchmarks in the native version of RODOS and compared them against the virtualized version. Fig. 4 presents the achieved results, corresponding to the normalized values of an average of 100 collected samples for each bench-

*Table 4. Correlation between guest-switching rate and performance*

|  | ***Performance*** |  |  |  |  |  |
|---|---|---|---|---|---|---|
| Rate (ms) | 1 | 2 | 5 | 10 | 50 | 100 |
| Perf. (%) | 91.7 | 95.7 | 98.3 | 99.1 | 99.8 | 99.9 |

mark. From the experiments it is clear that the virtualized version of RODOS only presents a very small performance degradation when compared with its native execution, i.e., <1%.

In the second part of the experiment, we evaluated how the guest-switching rate correlates with guest performance. To measure the influence of guest-switching rate in the performance loss, we repeated the experiments for a rate within a time window between 1 millisecond to 100 milliseconds. Tab. 4 shows the achieved results, where each column corresponds to the average performance of the measured results for the 6 benchmarks. As it can be seen, the performance of the virtualized RODOS range from 91.7% to 99.9%, respectively.

## 6. CONCLUSION

Complexity of modern safety-critical systems is growing at a frenetic rate. To accompany this trend, aeronautics and space industries are shifting from full federated architectures to an IMA approach. Virtualization technology has been used as an implementation technique to provide STP, but existent virtualization solutions for space are not ready to fully tackle the upcoming challenges of aerospace industry.

This paper describes a real time hypervisor for space applications assisted by COTS hardware. The system was deployed on a commercial Xilinx ZC702 board, demonstrating how it is possible to host an arbitrary number of guests on the non-secure world side of TrustZone-enabled processors. The secure hypervisor is flexible enough to run unmodified guest OSes at higher performance, while guaranteeing strong isolation between guests. Our evaluation demonstrated virtualized OSes run with more than 99% performance for a 10 milliseconds guest-switching rate. The reduced TCB size of RTZVisor decreases also effort for certification. Our solution makes use of several technologies needed for the demanding challenges of future aerospace applications: secure virtualization deployed under hybrid platforms.

Research roadmap will focus on three main directions. First, we will explore Xilinx Zynq Ultrascale+ to extend our solution for hybrid HMP. HMP are considered as the next generation multicore platforms, and they are being seen as the only path for adequate functionality consolidation and timing predictability. From another perspective, we will also explore the TrustZone technology of the new (ARMv8) Cortex-M and Cortex-R processors series to develop virtualization solutions for middle and low-end space applications. With the current trend of small satellites going viral, this technology can potentially impact the way those satellites are being built. Finally, and essentially because complexity is becoming impracticable to be manually managed, another part of our investigation will tackle the development of mechanisms and tools to assist design automation. We have already some on-going experiments around the development of Domain Specific Languages (DSL) to help in the configuration, customization and code generation of RTZVisor.

## ACKNOWLEDGMENTS

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT - Fundação para a Ciência e Tecnologia - (grant SFRH/BD/91530/2012 and UID/CEC/00319/2013), and also by Space Agency of the German Aerospace Center (DLR) with federal funds of the German Federal Ministry of Economics and Technology (BMWi) under 50RM1203.

## REFERENCES

[1] Abella J. et al., Towards Improved Survivability in Safety-critical Systems, Proceedings of the 17th IEEE International On-Line Testing Symposium (IOLTS), Athens, Greece, 2011.

[2] RTCS, DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, Washington DC, USA, 2005.

[3] Diniz N. et al., ARINC 653 in Space, Proceedings of the Data Systems in Aerospace (DASIA), Edinburgh, Scotland, 2005.

[4] Windsor J. et al, Time and space partitioning in spacecraft avionics, Proceedings of the 3rd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT), California, USA, 2009.

[5] Masmano, M. et al., An overview of the XtratuM nanokernel, Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Palma de Mallorca, Spain, 2005.

[6] Crespo A. et al., XtratuM an Open Source Hypervisor for TSP Embedded Systems in Aerospace, Proceedings of the Data Systems in Aerospace (DASIA), Istanbul, Turkey, 2009.

[7] Crespo, A. et al., Multicore partitioned systems based on hypervisor, Preprints of the 19th World Congress - The International Federation of Automatic Control, Cape Town, South Africa, 2014.

[8] VanderLeest S.H., ARINC 653 hypervisor, Proceedings of the 29th IEEE/AIAA Digital Avionics Systems Conference (DASC), Utah, USA, 2010.

[9] VanderLeest, S.H. et al., A safe & secure arinc 653 hypervisor, Proceedings of the 32nd IEEE/AIAA Digital Avionics Systems Conference (DASC), New York, USA, 2013.

[10] Joe H. et al., Full virtualizing micro hypervisor for spacecraft flight computer, Proceedings of the 31st IEEE/AIAA Digital Avionics Systems Conference (DASC), Virginia, USA, 2012.

[11] Tavares A. et al., RodosVisor - an Object-Oriented and Customizable Hypervisor: The CPU Virtualization, Embedded Systems, Computational Intelligence and Telematics in Control, No. 1, 2012.

[12] Tavares A. et al., Rodosvisor - An ARINC 653 quasi-compliant hypervisor: CPU, memory and I/O virtualization, Proceedings of the 17th IEEE Conference on Emerging Technologies & Factory Automation (ETFA), Krakow, Poland, 2012.

[13] Cereia M. et al., Virtual machines for distributed real-time systems, Computer Standards & Interfaces 31.1, 2009.

[14] Sangorrin D. et al., Dual operating system architecture for real-time embedded systems, Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Brussels, Belgium, 2010.

[15] Varanasi P. et al., Hardware-supported virtualization on ARM, Proceedings of the Second Asia-Pacific Workshop on Systems, Shanghai, China, 2011.

[16] Pinto S. et al., Towards a Lightweight Embedded Virtualization Architecture Exploiting ARM TrustZone, Proceedings of the 20th IEEE Conference on in Emerging Technologies & Factory Automation (ETFA), Barcelona, Spain, 2014.

[17] Moratelli C. et al., Full-Virtualization on MIPS-based MPSOCs embedded platforms with real-time support, Proceedings of the 27th Symposium on Integrated Circuits and Systems Design (SBCCI), Aracaju, Brazil, 2014.

[18] ARM, ARM Security Technology - Building a Secure System using TrustZone Technology, Technical Report PRD29-GENC-009492C, 2009.

[19] VanderLeest S.H. et al., MPSoC hypervisor: The safe & secure future of avionics, Proceedings of the 34th IEEE/AIAA Digital Avionics Systems Conference (DASC), Prague, Czech Republic, 2015.

[20] Gu, Z. et al., A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization, Journal of Software Engineering and Applications, pp. 277-290, 2012.