HYPERVISOR FOR REAL TIME SPACE APPLICATIONS

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

[1]Centro Algoritmi, University of Minho, Portugal,
+351 253 510180, sandro.pinto@dei.uminho.pt,atavares@dei.uminho.pt
[2]Aerospace Information and Technology, University of Würzburg, Germany,
+49 931 31 83715, sergio.montenegro@uni-wuerzburg.de

ABSTRACT

Virtualization technology is a mainstream tool in servers, presenting huge benefits in terms of power management and service consolidation. Over the last few years, virtualization started gaining also momentum in safety-critical systems, and may become a key technology for future space applications due to the reduction on size, weight and power budget while offering increased dependability.
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 today's systems lead aerospace industry to follow another strategy. European Space Agency proposed Integrated Modular Avionics for Space to be considered as reference flight computer architecture for space applications. Current approach goes towards higher integration provided by time and space partitioning of system virtualization. Identified issues and challenges go behind real time virtualization and penalties incurred by existent software-based virtualization solutions.
This work presents RTZVisor, a real time hypervisor for space applications assisted by commercial off-the-shell hardware. ARM TrustZone technology is exploited to implement a virtualization solution with low overhead and low footprint. This is demonstrated by running multiple partitions of RODOS operating system on a Xilinx ZC702 board.

1 MOTIVATION

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 had evolved in such a way that fully federated architectures, where each function is implemented in its own embedded controller, become completely impracticable. Naturally, those industries rapidly tried to find other alternatives, and aeronautics pioneering the shift from traditional federated architectures to an IMA [2] architecture (Figure 1). By combining several applications into one powerful computing resource, they were able to get a reduction on Size, Weight and Power (SWaP) of their computing systems, and consequently reduce the cost of their solutions. For example, Honeywell found that for their work on aircraft Airbus A380, IMA provided a 50% volume and 40% weight reduction from previous federated surveillance equipment [3].
As space domain shares the same basic needs of aeronautics, they rapidly concluded that IMA strategy could be spun-in to the space domain. The trend on the first European space projects

following an IMA approach was to integrate all separate data management and control system units into one single computing board, but without virtualized partitions [4]. Typically, a single real time operating system (RTOS) was in charge of managing all system resources and consolidating applications into different (application) partitions, being responsible for the successful coexistence of all software. The problem of such a consolidation strategy is that space applications have different levels of criticality, which mean a simple modification in one application results in a new certification and validation campaign at the maximum level of criticality. The introduction of Time and Space Partitioning (TSP) [5, 6] 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, TSP 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. Based on those concepts, European Space Agency (ESA) proposed Integrated Modular Avionics for Space (IMA-SP) as a reference flight computer architecture for space applications [7].

Virtualization technology has been used as an implementation technique to provide TSP. Over the last few years several works have been proposed in the aerospace domain [6, 8, 9, 10, 11]. Some of them follow a (software-based) full-virtualization approach, while others implement 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. Taking in mind the penalties incurred by software-based virtualization, research and industry focus their attention in providing hardware support to assist virtualization. While Intel, ARM and Imagination/MIPS introduced their own commercial off-the-shell (COTS) technologies, some researchers developed their own customized hardware [12]. Since then, several hardware-based solutions have been proposed [13, 14, 15], but none of them was specifically designed for the aerospace domain. Among existent COTS virtualization technologies, ARM TrustZone [16] is gaining particular attention due to the ubiquitous presence of ARM-based devices in the embedded sector, as well as the supremacy of TrustZone-enabled processors when comparing with Virtualization-enabled processors. The problem is existent TrustZone-based solutions [14, 17, 18] 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 virtualization solution assisted by COTS hardware carefully designed for real time space applications. A distinctive aspect of our hypervisor is the use of ARM TrustZone technology to assist virtualization, which allows to run multiple instances of an RTOS on a single on-board computer with really low overhead and low footprint. This is demonstrated by running several RODOS OS partitions on a Xilinx ZC702 board. Moreover, the recent announcement of ARM about their decision of introducing TrustZone technology in all Cortex-M and Cortex-R processors series altogether with the recent publication of Steven VanderLeest [19] envisioning how to tackle the challenges of the future of avionics, completely pushed our work to another level: we will be able to use the same virtualization approach for middle and low-end embedded applications, which can perfectly align with the trend of small satellites going viral; we have now an implemented prototype covering a huge part of the challenges that Steven envision, and a system platform that can be easily extended to cope with requirements like security and obsolescence.

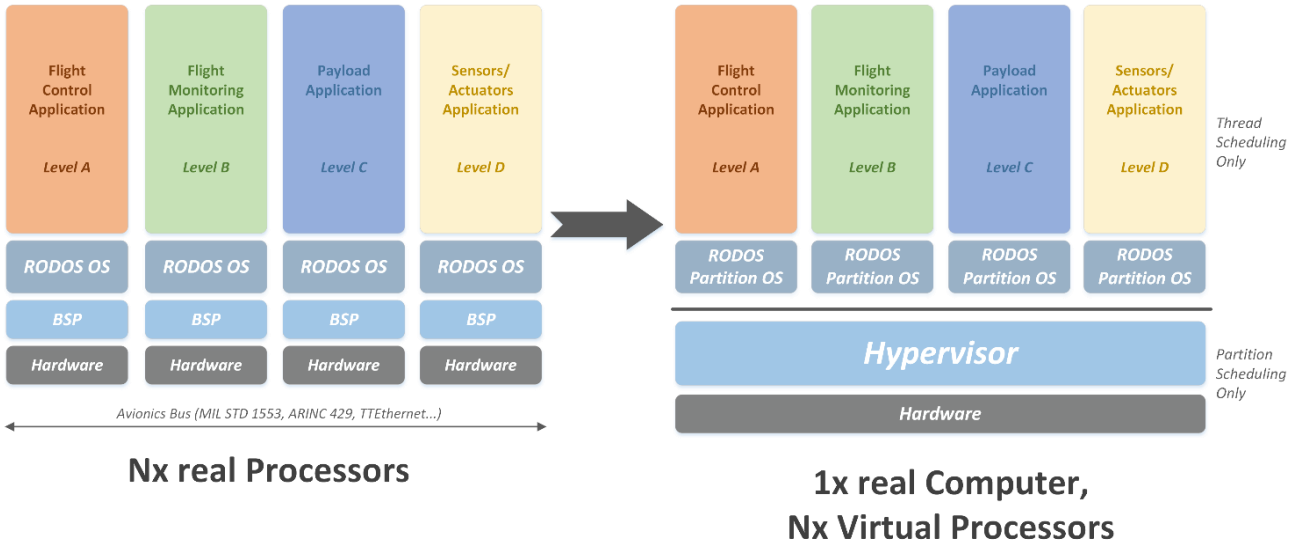

Figure 1 - Moving from fully federated architectures to an IMA approach

2 VIRTUALIZATION

Virtualization technology, which allows the co-existence of multiple OSes on the same hardware platform, is well established in the enterprise and cloud computing space, presenting huge benefits in terms of load balancing, power management and service consolidation. Over the last few years, virtualization has proven to be a gamer-changer in the embedded systems field [20]. The possibility of co-existence of heterogeneous OS environments altogether with the possibility of isolation of multiple workloads, with different levels of criticality, have attracted embedded industries (automotive, aerospace, medical, etc) to build systems with smaller form factor and reduced bill of materials (BOM).

Traditional embedded virtualization solutions [6, 8, 9, 10, 11] follow essentially two different approaches: full-virtualization and paravirtualization. In full-virtualization [8, 9, 10, 11] partition OSes do not require any modification into the kernel code, but the hypervisor needs to trap and emulate all the privileged instructions, incurring in a significant performance degradation. Paravirtualization [6, 8, 11], in contrast, requires modification of partition OSes to include specific hypercalls into the kernel, to request services directly from the hypervisor. While providing several performance advantages, paravirtualization incurs in a higher design cost since each guest OS must be modified to fit the hypervisor. More recently, taking in mind the penalties incurred by software-based virtualization approaches, research and industry focus their attention in providing hardware support to assist virtualization. The Big Players of processors industry introduced their own COTS technologies: Intel introduced Intel Virtualization Technology (VT), Intel Trusted Execution Technology (TXT) and Intel Software Guard Extensions (SGX); ARM presented ARM Virtualization Extensions (VE) and ARM TrustZone; and, finally, Imagination/MIPS recently announced OmniShield.

Intel VT and ARM VE allow the implementation of hardware-based full-virtualization [13, 21]. Both technologies rely on a new privileged processor mode - the hypervisor mode - altogether with MMU support for 2-level address translations. This features allow for direct execution of guests without the need for de-privileging, enabling certain instruction to directly affect virtual registers instead of trapping to the hypervisor: the minimization of the number of trap-and-emulate occurrences boosts

considerably the execution performance. ARM TrustZone technology, although implemented for security purposes, allow a special kind of virtualization similar to full-virtualization. With a virtual hardware support for dual world execution, a new privileged processor mode called monitor mode, and other TrustZone features like memory segmentation, it is possible to provide time and spatial isolation between execution environments. This technology starts being widely used for a dual-OS configuration [14, 17, 18, 22], but there is a lack of solutions for multiple guest OS support. MIPS Virtualization, powered by Imagination OmniShield technology, not only adds support for hardware full-virtualization, but also extends the binary approach of TrustZone Technology to create multiple secure domains: applications that need to be secure are effectively and reliably isolated from each other, as well as protected from non-secure applications. The hypervisor developed by Hessel research group [15, 23] is an example of the use of this technology to implement real time virtualization. Finally, regarding Intel TXT and Intel SGX, we are not aware of any work that exploits those technologies to implement, per se, any kind of virtualization. This technologies, contrarily to TrustZone, are just being used to add security features to existent Intel-based virtualization solutions.

3 TRUSTZONE OVERVIEW

TrustZone technology [16] refers to security extensions implemented by ARM since ARMv6 architecture. The TrustZone hardware architecture can be seen as a dual-virtual system, partitioning all system physical resources into two completely separated execution environments: the Secure and the Non-Secure worlds.

3.1 Processor

At the processor level, the most significant architectural change is its partition into two separate worlds - the secure world and the non-secure world. A new 33rd processor bit, the NS (Non-Secure) bit accessible through the SCR (*Secure Configuration Register*) register, indicates in which world the processor is currently executing, and is propagated over the memory and I/O peripherals buses. To preserve the processor state during the world switch, TrustZone adds an extra processor mode: the monitor mode. The monitor mode is completely different from other supported modes, because independently of the state of NS bit, when the processor runs in this mode the state is always considered secure. As the processor only runs in one world at a time, software stacks in the two worlds can be bridged via a new privileged instruction - SMC (*Secure Monitor Call*). The monitor mode can also be entered by configuring it to handle IRQ, FIQ, and Aborts exceptions in the secure world. To provide the exception behavior described above, TrustZone specifies three sets of exception vector tables - one for the normal world, one for the secure world, and another for the monitor mode. Furthermore, to guarantee a strong isolation between secure and normal states, some special registers are banked, such as a number of System Control Coprocessor (CP15) registers. Some secure critical processor core bits and CP15 registers are either totally unavailable to non-secure world or access permissions are closely under supervision of the secure world.

3.2 Memory

TrustZone extensions split CPU into two distinct environments, which, per se, is not enough to fully partition memory between two worlds. The TrustZone Address Space Controller (TZASC) and the TrustZone Memory Adapter (TZMA) extend security to protect multiple regions of memory from software attacks. TZASC enables partition of DRAM into different memory regions: the TZASC has a programming interface - accessible only from the secure side - that can be used to configure a

specific memory region as secure or non-secure. By default, secure world applications can access normal world memory but the reverse is not possible. TZMA provides similar functionality but for off-chip ROM or SRAM.
The TrustZone-aware Memory Management Unit (MMUs) provides two distinct MMU interfaces, enabling each world to have a local set of virtual-to-physical memory address translation tables. The isolation is still available at the cache-level, because processor caches have been extended with an additional tag which signals in which state the processor accesses the memory.

3.3 Devices and Interrupts

System devices can be dynamically configured as secure or non-secure through the TrustZone Protection Controller (TZPC), opening possibility to change the security state of a device at run-time.
To support the robust management of secure and non-secure interrupts, the Generic Interrupt Controller (GIC) provides both secure and non-secure prioritized interrupt sources. An interrupt can be configured as a secure interrupt through the *Interrupt Security Register*. In addition, the interrupt controller allows prioritization of interrupts, allowing the configuration of secure interrupts with a higher priority than the non-secure interrupts. Such configurability prevents non-secure software to perform a denial-of-service attack against the secure side. Besides that, the GIC supports several interrupt models, which allows the configuration of IRQs and FIQs to secure or non-secure interrupt sources. The suggested model by ARM proposes the use of IRQs as non-secure world interrupt sources, and FIQs as secure interrupt sources.

4 RODOS

Real time On-board Dependable Operating System (RODOS) [24] was originally developed for space applications at DLR (German space agency), and now distributed as open source. RODOS was designed for application domains demanding high dependability (e.g., space) and targets the irreducible complexity in all implemented functions.
An important aspect in the selection of RODOS is its integrated real time middleware. Developing the control and payload software on the top of a middleware provides the maximum of modularity today. Applications/modules can be developed independently and it is very simple to interchange modules without worrying about side effects, because all modules are encapsulated as Building Blocks (BB) and they can access other resources only by well-defined interfaces.
RODOS was implemented as a software framework in C++. It is organized in layers: the lowest layer (1) is responsible for managing the embedded system hardware (HAL: Hardware Abstraction Layer); the next layer (2), kernel, administrates the local resources, threads and time. On top of the kernel is located the middleware (layer 3) which enables communication between BBs using a publisher subscriber multicast protocol. Finally on the top of the middleware the user may implement his applications (layer 4) as a distributed software network of simple BBs. The Building Blocks API on the top of the middleware follows a service oriented interface. BBs interact by providing services to other BBs and using services from other BBs.
As mentioned before, the original purpose of RODOS was to control satellites. It was designed as the brain of the Avionic system and introduces the NetworkCentric concept [25]. A NetworkCentric core avionics machine consists of several harmonized components which work together to implement dependable computing in a simple way. In a NetworkCentric system we have a software network of BBs and a hardware Network interconnecting vehicles (radio communication), computers inside of

vehicles (buses and point to point links), intelligent devices (attached to buses) and simple devices attached to front-end computers. To communicate with (node) external units, including devices and other computing units, each node provides a gateway to the network and around the network's several devices may be attached to the system. The messages exchange service provided by the middleware and gateways is asynchronous, using the publisher-subscriber protocol. No fixed communication paths are established and the system can be reconfigured easily at run-time. For instance, several replicas of the same software can run in different nodes and publish the result using the same topic, without knowing each other. A voter may subscribe to that topic and vote on the correct result. Application can migrate from node to node or even to other vehicles without having to reconfigure the communication system. The core of the middleware distributes messages only locally, but using the integrated gateways to the NetworkCentric network, messages can reach any node and application in the network. The communication in the whole system includes software applications, computing nodes and even IO devices. Publishers make messages public under a given topic. Subscribers (zero, one or more) to a given topic get all messages which are published under such topic.

5 RTZVISOR

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 system partitions. All data structures and hardware resources are pre-defined and configured at design time, and devices and interrupts can be directly managed by specific partitions. Exceptions and errors are managed through a special component called Health Monitor, which is able to recover partitions from undefined states.
Figure 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 partition runs in the non-secure world side, while inactive partitions are preserved in the secure world side; for active partitions the RTOS runs in the kernel mode, while RT applications run in user mode.

5.1 Virtual CPU

TrustZone technology virtualizes each physical CPU into two virtual CPUs: one for the secure world and another 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 partition OSes, all of them need to run in 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 partition 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, partition and real CPU is supported.

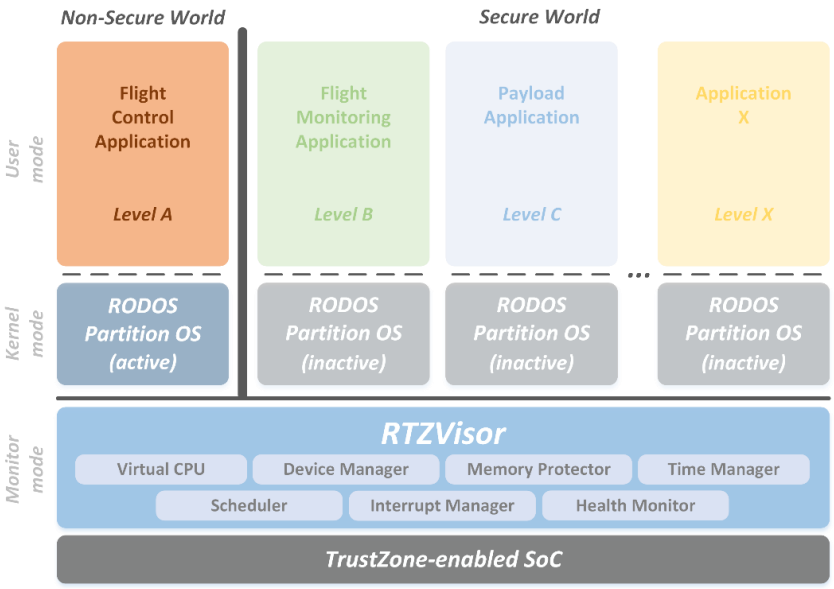

Figure 2 - System Architecture

5.2 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 partition OSes, and also to implement isolation between partitions.

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 partitions, basically by dynamically changing the security state of the memory segments of partitions. Only the partition that is currently running (in the non-secure side) should have its own(s) memory segment(s) configured as non-secure, and the remaining memory as secure. If the running partition tries to access a secure memory region (belonging to an inactive 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 partitions (belonging momentously to the secure side) to change the state of another partition.

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 partitions (one memory segment is for the hypervisor itself). Our system relies on the TZASC to implement isolation between partitions, 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 partition OS. Figure 3 depicts the memory setup and respective secure/non-secure mappings, for a virtualized system consisting in the hypervisor and four partitions. In this specific configuration, the hypervisor uses the first memory segment (0x00000000 – 0x03FFFFFF), and has access to all memory. Partition OS-0 uses the third 64MB memory segment, and is only allowed to access one non-secure memory segment (0x08000000 – 0x0BFFFFFF); Partition OS-1/2/x are mapped the same way, but within their respective memory segment.

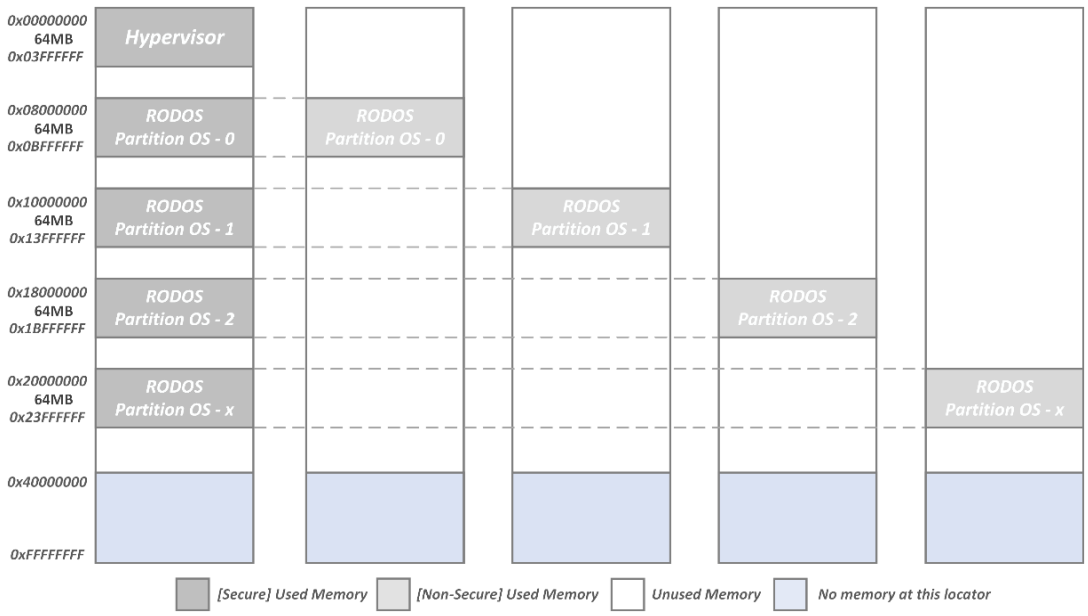

Figure 3 – System memory map

5.3 Scheduler

RTZVisor implements a cyclic scheduling policy, to ensure one partition cannot use the processor for longer than its defined CPU quantum. The time of each slot can be different for each partition, depending on partition 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 partition is only interrupted when the complete slot is over.

5.4 Devices

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

RTZVisor implements device virtualization adopting a pass-through policy, which means devices are managed directly by partitions. To ensure strong isolation between partitions, devices are not shared between them and are assigned to respective partitions at design time. To achieve this strong isolation at device level, devices assigned to partitions are dynamically configured as non-secure or secure, depending on partition state (active or inactive). This guarantees an active partition cannot compromise the state of a device belonging to another partition, and if an active 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 accessed by any partition.

5.5 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 models of configuration 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 partition is under execution, only the interrupts managed by this partition are enabled, which minimizes inter-partition interferences through hardware. Interrupts of inactive partitions are momentaneously configured as secure, and consequently redirected to the hypervisor. The hypervisor receives and processes the interrupt, and the virtual GIC of the inactive guest is updated. When a partition is rescheduled, interrupt is then processed. The prioritization of secure interrupts avoid active partition to perform a denial-of-service attack against the secure side (hypervisor).

5.6 Time

Temporal isolation in virtualized systems is typically achieved using two levels of timing: the hypervisor level and the partition level. For the partition level, hypervisors typically provide timing services which allow guests to have notion of virtual or real time. In the first case, each time a partition is inactive the time is paused, and once the guest is rescheduled the timekeeping is resumed. In the meantime, there is a drift between the guest time and the absolute time. For mission critical real time systems, where time-responsiveness plays a critical role, this is an undesired approach, because partitions have necessarily to keep track of the wall-clock time.

RTZVisor implements also two levels of timing: it has internal clocks for managing the hypervisor time, and internal clocks for managing the 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 partition. This means that despite of what is happening in the active partition, 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 partition are directly managed and updated by the partition 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 partition is rescheduled, its internal clocks and related data structures are updated with the time elapsed since its previous execution.

5.7 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. For example, if a partition OS tries to access a portion of memory outside its boundaries, the hypervisor detects the space violation and immediately reboots the partition.

6 EVALUATION

RTZVisor was evaluated on a Xilinx ZC702 evaluation 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. Our evaluation focused on the following metrics:

- Memory footprint: amount of memory (bytes) required by RTZVisor;
- Partition context switch: time needed by the hypervisor to switch between partitions;
- Partition performance loss: measured overhead introduced by RTZVisor at partition level.

To evaluate partition context-switch time and performance loss we specified two different test case scenarios:

1. **Test 1** - MMU, data and instruction cache and branch prediction (BP) support for partitions were disabled;
2. **Test 2** - MMU, data and instruction cache and branch prediction support for partitions were enabled;

In all test scenarios RTZVisor and all RODOS OS partitions were compiled using the ARM Xilinx toolchain, and compilation optimizations were disabled.

6.1 Memory footprint

To access memory footprint results we used the size tool of ARM Xilinx Toolchain. Table 1 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; (ii) and the careful design and static configuration of each hypervisor component.

Table 1 – Memory footprint results (bytes)

	.text	.data	.bss	Total
RTZVisor	5568	192	0	5760

6.2 Partition context switch

To evaluate the partition 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.

Table 2 – Context-switch evaluation (microseconds)

Context-switch operation	Time – Test 1 (µs)	Time – Test 2 (µs)
1. Timer interrupt management	1.620	1.625
2. Save vCore context	1.873	1.867
3. MMU and cache management	---	184.417
4. Scheduler	4.000	4.003
5. vCP15 context-switch	---	4.890
6. vGIC context-switch	31.533	31.542
7. Time management	53.033	52.985
8. Memory configuration	1.053	1.052
9. Restore vCore context	1.963	1.963
TOTAL	**95.075**	**284.344**

The list of activities as well as the measured time for each test case scenario are presented in Table 2. As it can be seen, in the first test case scenario (Test 1), the activities which present higher consuming time are the virtual GIC context-switch and the time management. In both cases, there are a chance to optimize both operations, because our current solution is more focused on generalization instead

of particularization. In the second test case scenario (Test 2), since MMU and cache support for guest OSes are needed, the major source of overhead (approx.184µs) is related with the MMU and cache management. On this case, there is also a chance to optimize this operation, based on some existent works that focus on cache optimization for virtualization.

6.3 Partition performance loss

The Thread-Metric Benchmark Suite consists in 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 (CS); preemptive scheduling (PS); interrupt processing (IP); interrupt preemption processing (IPP); synchronization processing (SP); message processing (MP); and memory allocation (MA). For each benchmark the score represents the RTOS impact on the running application, where higher scores correspond to a smaller impact.

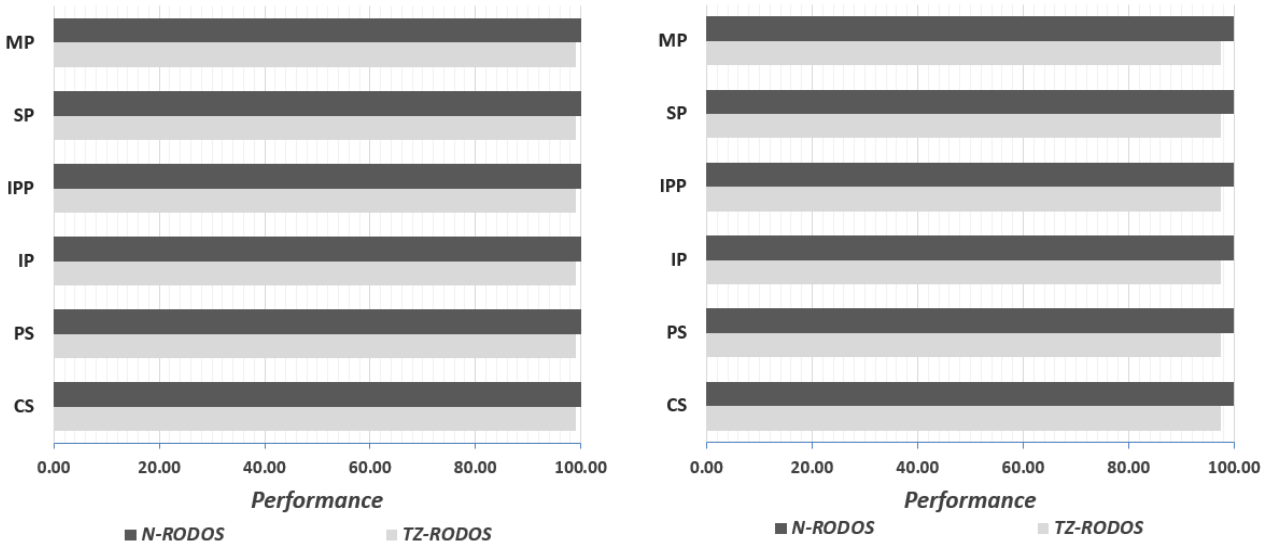

Figure 4 – Thread Metrics Benchmark Results: Test 1 (left) and Test 2 (right)

For the first part of the experiment RTZVisor was configured with a 10 milliseconds (ms) partition-switching rate. The system was set to run one single partition, and the hypervisor scheduler was forced to reschedule the same partition, 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. Figure 4 presents the achieved results, corresponding to the normalized values of an average of 100 collected samples for each benchmark. In both test case scenarios – Test 1 (left) and Test 2 (right) -, it is clear that the virtualized version of RODOS only presents a very small performance degradation when compared with its native execution - <1% and <3%, respectively. As expected for what was stated in section 6.2, in Test 1 – Figure 4 (left) - the performance degradation is smaller, because the guest-switching operation does not require to save the state of the CP15, as well as performing some MMU and cache related operations.

In the second part of the experiment we evaluated how the partition-switching rate correlates with partition performance loss. To measure the influence of partition-switching rate in the performance loss, we repeated the experiments for a rate within a time window between 1 millisecond to 1 second. Table 3 shows the achieved results, where each line 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 91.70% to 99.98% and 73.95% to 99.96% for Test 1 and Test 2, respectively. For the

second test case scenario, the significant performance degradation above 5 milliseconds is mainly explained by two reasons: first, as aforementioned, when MMU and caches are enabled, the list of internal activities of context-switch operation is higher; and secondly, since caches have to be cleaned and invalidated each time a partition is reschedule, partitions will not take advantage of them until they are filled.

Table 3 – Correlation between partition-switching rate and performance loss

Context-switch tick (ms)	Performance - Test 1 (%)	Performance - Test 2 (%)
1000	99.98	99.96
100	99.91	99.73
50	99.83	99.46
10	99.13	97.37
5	98.30	94.75
2	95.76	86.96
1	91.70	73.95

7 CONCLUSION

Complexity of modern safety-critical systems is growing at a frenetic rate. To accompany this trend, aeronautics and space industries are moving from full federated architectures to an IMA approach. Virtualization technology has been used as an implementation technique to provide time and space partitioning, but existent virtualization solutions fail in guaranteeing simultaneously flexibility and performance.

RTZVisor is a real time hypervisor for space applications assisted by COTS technology (ARM TrustZone). It was deployed on a commercial Xilinx ZC702 board, demonstrating how it is possible to host an arbitrary number of partition OSes on the non-secure world side of TrustZone-enabled processors. The hypervisor is flexible enough to run unmodified guest OSes at higher performance. Our evaluation demonstrated virtualized OSes run with more than 99% performance for a 10 milliseconds partition-switching rate. The reduced TCB size of RTZVisor decreases also effort for certification. The distinctive aspect of our work is the use of ARM TrustZone technology as a foundation for hardware-based virtualization for real time space applications. Furthermore, our solution makes use of all technologies needed for what challenges of future aerospace applications are demanding: secure virtualization deployed under hybrid platforms.

Future work will mainly focus on the development of an inter-domain message passing schema, and on the extension for multicore architectures. Generalization of current solution for future ARM architectures, as well as an investigation of an hardware-software co-designed approach are also under scope.

9 ACKNOWLEDGEMENT

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.

10 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] Ramsey J., *Integrated Modular Avionics: Less is More*, Avionics Today, 1 Feb 2007. Online: http://www.aviationtoday.com/av/commercial/Integrated-Modular-Avionics-Less-is-More_8420.html

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

[5] 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.

[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] Windsor J. et al, *Integrated modular avionics for spacecraft - User requirements, architecture and role definition*, Proceedings of the 30th IEEE/AIAA Digital Avionics Systems Conference (DASC), Seattle, USA, 2011.

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

[9] Han S. et al., *Full virtualization based ARINC 653 partitioning*, Proceedings of the 30th IEEE/AIAA Digital Avionics Systems Conference (DASC), Seattle, USA, 2011.

[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 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.

[12] Garcia P. et al., *Towards hardware embedded virtualization technology: architectural enhancements to an ARM SoC*, ACM SIGBED, p. 45-47, 2014.

[13] Liu C.-t. et al., *CASL hypervisor and its virtualization platform*, Proceedings of the 2013 IEEE International Symposium on Circuits and Systems (ISCAS), Beijing, China, 2013.

[14] 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.

[15] 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.

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

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

[18] 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.

[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] Heiser G., *Virtualizing embedded systems - why bother?*, Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC), New York, USA, 2011.

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

[22] Frenzel T. et al., *ARM TrustZone as a virtualization technique in embedded systems*, Proceedings of Twelfth Real-Time Linux Workshop, Nairobi, Kenya. 2010.

[23] Zampiva S. et al., *A hypervisor approach with real-time support to the MIPS M5150 processor*, Proceedings of the 16th IEEE International Symposium on Quality Electronic Design (ISQED), Santa Clara, USA, 2015.

[24] Montenegro S. et al., *RODOS - Real Time Kernel Design for Dependability*, ESA Special Publication, Vol. 669, 2009.

[25] Montenegro S., *Network Centric Core Avionics*, Proceedings of the 1st IEEE International Conference on Advances in Satellite and Space Communications, Colmar, France, 2009.