Towards a Lightweight Embedded Virtualization Architecture Exploiting ARM TrustZone

S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral and A. Tavares
Centro Algoritmi - University of Minho
{sandro.pinto, d.oliveira, j.pereira, nuno.cardoso, jorge.cabral, adriano.tavares}@algoritmi.uminho.pt, mongkol@ait.ac.th

Abstract—**Virtualization has been used as the *de facto* technology to allow multiple operating systems (virtual machines) to run on top of the same hardware platform. In the embedded systems domain, virtualization research has focused on the coexistence of real-time requirements with non-real-time characteristics. However, existent standard software-based virtualization solutions have been shown to negatively impact the overall system, especially in performance, memory footprint and determinism.**

This work in progress paper presents the implementation of an embedded virtualization architecture through commodity hardware. ARM TrustZone technology is exploited to implement a lightweight virtualization solution with low overhead and high determinism, corroborated by promising preliminary results. Research roadmap is also pointed and discussed.

Index Terms—**Virtualization, TrustZone, Monitor, Real-Time Embedded Systems, ARM.**

I. Introduction

Virtualization technology enables concurrent execution of multiple VMs (Virtual Machines) on the same hardware (single or multicore) processor, allowing the co-existence of multiple OSes (Operating Systems) environments on a single physical platform [1]. Traditionally, it has been used in enterprise and cloud computing space (server environments) to maximize resource usage and availability [2], and, over the last few years, this technology reached the embedded systems field [3], [4]. With the emergent complexity of modern embedded devices, which increasingly ask for general purpose computing characteristics while still constrained by real-time requirements, embedded virtualization has emerged as a solution to address the aforementioned concerns.

Typical existent embedded virtualization solutions [5], [6], [7], [8] follow essentially two different implementations: full-virtualization and paravirtualization [9]. With full-virtualization, guest OSes are supported without any modification. The hypervisor or VMM (Virtual Machine Monitor) needs to trap and emulate privileged instructions, which leads to a significant performance degradation. Oppositely, with paravirtualization, the guest OS is modified to include specialized system calls (hypercalls) into the kernel, that enables it to request services directly from the hypervisor. With this static approach, the execution performance increases significantly, but the engineering and maintenance effort associated with custom modifications lead to lower productivity and longer time-to-market.

More recently, taking in mind the rigid constraints (e.g., performance, memory, power, safety, security) of the embedded domain, research has focused on the development of efficient embedded virtualization solutions, leveraging hardware assistance for virtualization [10], [11]. Intel, ARM and AMD Virtualization Technologies, altogether with ARM TrustZone and Intel TxT (Trusted Execution Technology) are examples of existent technologies that can be exploited to implement efficient and secure embedded virtualization solutions. Driven by the ubiquitous presence of ARM-devices in the embedded market as well as the supremacy of ARM TrustZone-based SoCs (System-on-a-Chip) to ARM SoCs with Virtualization Extensions, our work and research is focusing on the exploration of the first mentioned technology.

This work in progress paper presents an implementation of a TrustZone-based virtualization architecture, which allows the execution of a GPOS (General Purpose Operating System) - Linux - side by side with a RTOS (Real-Time Operating System) - FreeRTOS. Since the presented implementation is assisted by commodity hardware, performance overhead and memory footprint are small. Preliminary results corroborate the viability of the presented implementation.

II. ARM TrustZone Overview

TrustZone technology [12] refers to security extensions implemented by ARM in modern applications processor cores, including the ARM1176, Cortex-A5/A7/A8/A9/A15, and the newest 64-bit Cortex-A53/A57. This hardware security extensions virtualizes a physical core as two virtual cores, providing two completely separated execution domains: the *secure world* for the security subsystem and the *non-secure world* for everything else. The state of the processor can be changed by enabling or disabling the NS (Non-Secure) bit of the SCR (*Secure Configuration Register*), exposed through coprocessor (CP15) interface [13].

To switch the processor between the secure and the non-secure world, a special new secure processor mode, called *monitor mode*, was introduced. 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. To enter the monitor mode, a new privileged instruction was also specified - SMC (*Secure Monitor Call*). Furthermore, some exceptions sources, as interrupts, can be also configured to be handled in monitor mode upon triggering. This forces the processor to enter monitor mode without using the dedicated privileged instruction. The template of the monitor code is defined by the

978-1-4799-4845-1/14/$31.00 ©2014 IEEE

developer, but it generally saves the state of the current world and restores the state of the world being switched to.

At the hardware-level, TrustZone is more than extensions build into the core to ensure a strong isolation between the two worlds. Additional secure components are also built-in the SoC, such as a secure boot ROM (Read-Only Memory) to configure the system, a secure RAM (Random-Access Memory) used to store and run trusted code (e.g. Digital Rights Management engines and payment agents), or a secure non-volatile or one-time programmable memory for storing master keys [14]. Apart of all of these secure extensions, the AXI (Advanced eXtensible Interface) system bus carries extra control signals to restrict access to the read and write channels on the main system bus. This feature enables the possibility of the TrustZone architecture to secure peripherals (e.g. interrupt controllers, timers, and user I/O devices).

III. TrustZone-based Virtualization Architecture

The idea of using TrustZone as a virtualization technique in embedded systems was first introduced by Frenzel et al. [15]. They discovered that TrustZone provides a specialized, hardware-based form of system virtualization, especially in the case of two VMs. Since the number of VMs corresponds exactly with the number of isolated states supported by the processor, a rich or multimedia operating system (e.g. Linux, Android) could run in non-secure world, while safety/security-critical software could run in the secure world. Since the monitor mode has full view of the processor, using that mode to implement the VMM removes the need to modify the OS hosted on the non-secure side. Despite running in privileged mode, GPOS continues being less privileged than VMM component, since it can not access the state of the secure side. Besides, TrustZone also decreases the performance overhead, eliminating one of its most important sources - privilege instruction emulation -, and speeding up the context switch between VMs. To achieve the latter, the architecture is endowed by an extensive number of banked processor and coprocessor registers.

A. Architecture Description

Fig.1 depicts the TrustZone-based embedded virtualization architecture. As it can be seen, there are three main software components: the RTOS (FreeRTOS); the GPOS (Linux); and the Virtual Machine Monitor (VMM). The GPOS, running in the normal world, provides a rich and flexible environment, useful for running man-machine interfaces as well as internet-based applications and services. On the other hand, the RTOS, running in the secure world, provides a real-time environment essential for the development of applications which need to guarantee specific deadlines. Finally, the VMM component, running also in the secure world but in monitor mode, is responsible for managing the Virtual Machine Control Block (VMCB) of each VM. When a VM can be executed by the physical processor, the VMM saves the current state of the virtual processor on the corresponding VMCB, and restores the VMCB of the other VM on the physical processor.

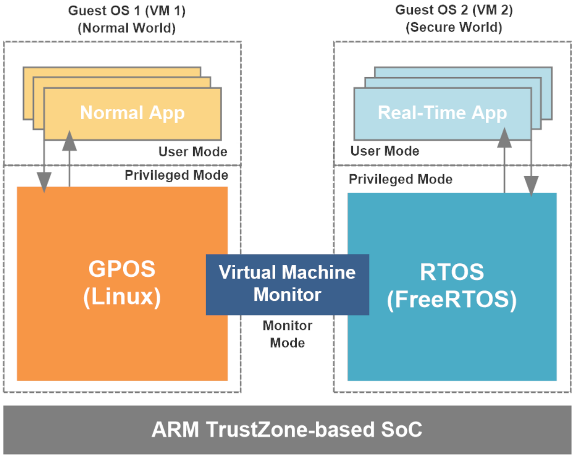

Fig. 1: Embedded Virtualization Architecture

B. Execution Flow

The system starts on the secure side with the start-up routine. This routine is responsible for a set of operations which includes registers initialization as well as stacks, memory, peripherals and interrupt controller configuration; e.g., an amount of memory is configured as secure and another as non-secure. The GIC (Generic Interrupt Controller) is also configured to route *fast interrupt requests* (FIQ) to secure world, and *interrupt requests* (IRQ) for non-secure world (Fig.2). On the SCR register the FIQ and IRQ bits are disabled to guarantee that FIQ/IRQ exceptions do not cause a switch to monitor mode, and consequently the VM switch is only performed through SMC instruction.

After the boot process, the RTOS - FreeRTOS in our implementation - starts scheduling its own tasks. When all the real-time tasks are blocked and/or suspended, the idle task performs a system call that is responsible for explicitly invoking the VMM, executing the SMC instruction (Fig.2). Immediately, the processor changes to the monitor mode and starts executing the VMM, jumping to the specific handler of the monitor vector table. Hence, the processor execution is routed to the SMC handler which prepares the transition to the non-secure world.

The next step performs the context-switch operation. Concretely, the processor state of the secure side (FreeRTOS) is saved in its own VMCB, and the VMCB of the non-secure side (Linux) is restored. An exception to this is in the very beginning (i.e., first time execution) when due to optimization purpose, only the processor state of the secure side is saved, the supervisor mode (SVC) is set, and the linker register is updated with the start address of the GPOS kernel. At the end, the VMM enables the FIQ and NS bits of SCR register and jumps to the initialized/restored non-secure world address.

As it can be noticed, until this moment no operation on cache was performed. In fact, TrustZone permits that cache

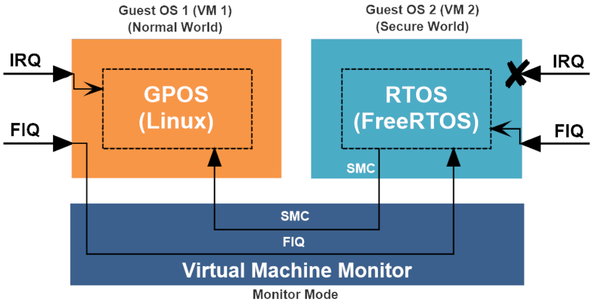

Fig. 2: Execution flow diagram

TABLE I: VMM performance statistics

	Execution Time (clock cycles)			
	min	*max*	μ	σ
Switch to NS world	2431	2568	2478	52.5
Switch to S world	2081	2245	2109	48.9

TABLE II: VMM memory statistics

	Memory Footprint (bytes)			
	.text	**.data**	**.bss**	**Total**
VMM	848	0	244	1092
FreeRTOS	17674	16	66000	83690
Linux	2874978	52	4120	2879150

entries of the secure world and the normal world co-exist together. TrustZone support in the cache controller specifies that a NS tag bit is attached to all data in the cache or in buffers. This support removes the need for a cache flush when switching between VMs, and contributes to a faster context-switch.

On the non-secure side, the GPOS will run until the moment that a FIQ is triggered (Fig.2). Since the FIQ bit had been previously enabled in SCR register, the arrival of a FIQ request brings the processor into monitor mode, jumping to the FIQ handler of the monitor vector table. At this moment, the VMM begins executing and prepares the context-switch operation. It starts by disabling the FIQ and NS bits of SCR register, saves the full processor state view of the non-secure side into its VMCB, acknowledges the FIQ request and restores the secure side context from the VMCB.

At this point, the processor returns to the RTOS kernel, which will start dispatching tasks again. The processor will stay in the secure world until the moment that the idle task is re-scheduled. When it happens, the processor performs all previously described steps again.

IV. Preliminary Results

The implemented virtualization architecture was tested on a Xilinx ZC702 board with a dual ARM Cortex-A9 running at 800MHz. In spite of using a multicore hardware architecture, our current implementation only supports a single-core configuration. Performance results were obtained by exploiting the PMU (Performance Monitoring Unit) component. Memory footprint results were collected using the size tool of ARM Xilinx toolchain.

In order to assess the overhead introduced by our VMM implementation, two context switch operations were performed. The selected scenarios encompass:

1) *Switch to non-secure world* - The VMM performs the context switch operation from the secure to the non-secure world, giving control to the Linux kernel. The number of clock cycles is measured from the exact moment of the SMC instruction on the secure side until the instant that the processor reaches the address of the non-secure side;
2) *Switch to secure world* - The VMM performs the context switch operation from the non-secure to the secure world, giving control to the FreeRTOS kernel. The number of clock cycles is measured from the exact moment of the FIQ exception on the non-secure side until the instant that the processor reaches the kernel running on the secure side.

Each test was repeated twenty times, and the results report the minimum (*min*), maximum (*max*) and mean (μ) value as well as the standard deviation (σ) of the collected measurements. Despite executing the same number of instructions by the VMM in each iteration, the number of clock cycles needed to execute the instructions varies due to the presence of dynamic architectural features (e.g., write buffer and caches) in the Cortex-A9 processor.

Table I presents the performance overhead introduced by the VMM. Considering that the clock frequency of the used processor is 800MHz, this corresponds to an average execution time of 3.10μs and 2.64μs to perform a full switch between secure to non-secure and non-secure to secure worlds, respectively. Considering that the measured average time necessary to perform a task switch in the FreeRTOS was 2.02μs, the VM context-switch only has an average overhead of 30.6% and 53.4% relatively to the FreeRTOS task context-switch. Furthermore, as was previously explained, the VMM runs with all interrupt sources disabled, thus the worst case scenario happens when a FIQ request arrives when a secure to non-secure context switch is starting. In this case, the request is handled only after two complete world switches, which represents a worst case interrupt latency of 6.02μs. Since the latency introduced by the VMM on the FIQ request handling has a deterministic upper bound - limiting the variance introduced by dynamic architectural features -, it can be taken into account when designing the real-time system.

Table II displays the memory footprint (bytes) of each software component of the implemented architecture. As it can be seen, the memory overhead introduced by the VMM is substantially small than the RTOS. Concretely, the total amount of memory required by the VMM is approximately 1.30% and 0.04% of the total amount of memory required by the FreeRTOS kernel and the Linux kernel (uncompressed vmlinux), respectively.

V. Research Roadmap

Work in the near future will focus on the extension of the current architecture to deal with shared devices (I/O). At this stage, each peripheral is dedicated to one world. So, each device is marked as secure or non-secure (hardware bit), and accessed only by the respective VM. However, the idea is to abolish this limitation and integrate a new shared device access mechanism in the current architecture. Shared peripherals will be considered always secure, and accesses by the GPOS will be monitored by the VMM. Since the RTOS is running on the secure side, it has direct and privileged access.

After, research will investigate TrustZone's ability to support an arbitrary number of VMs. In the current implementation the number of VM coincides exactly with the number of isolated states supported by the processor. Consequently, the VMM was fitted to explore this correlation between the number of VM and the number of virtual processors supported inherently by the processor architecture. Hence, the idea is to extend the current architecture, implementing a more generic and sophisticated VMM. Thus, it will be possible to manage more than two VMs. The subsequent step will encompass a multicore approach. After expanding the virtualization architecture for multi-guest support, it will also be redesigned for multicore support. Investigation will focus on symmetric multiprocessing (SMP) and the necessary changes which should be introduced on the extended virtualization architecture.

From a different perspective, research will continue towards the development of a TEE [16]. The current architecture will be slightly modified, to target critical applications that deal with sensitive information, as secure payments or content protected by Digital Rights Management. Adopting a service-client methodology, FreeRTOS will work as a secure OS that provides secure services for client applications running on the Linux side. TrustZone API (Application Programming Interface) will be implemented as a standard communication mechanism between both OSes. At the end, the TEE will be consolidated with the extended VMM, creating a complete framework targeting safety and security requirements. In this way it will be possible to provide an adaptable one-size-fits-all runtime environment for critical/secure applications by one side, and allow the consolidation of multiple OSes by the other side.

VI. Conclusion

On the last few years, the interest in solutions that use virtualization technology to consolidate multiple workloads has been increased, especially on the embedded systems field. Embedded Industrial applications, for example, need to guarantee the deadlines of real-time tasks, while at the same time, integrating rich environments for monitoring and network purposes. However, existent pure software-based virtualization solutions introduce significant performance and memory overhead or require significant engineering and maintenance effort.

This paper presented a work in progress towards the implementation of a TrustZone-based virtualization architecture. Exploiting ARM commodity hardware technology, a lightweight virtualization architecture was implemented on a commercial Xilinx ZC702 board, showing how a GPOS (Linux) can coexist with a RTOS (FreeRTOS) with low performance overhead and memory footprint. As demonstrated by preliminary results, the performance and memory overhead introduced by the VMM are very small. Concretely, the measured VM context-switch overhead was 3.10μs to switch from the RTOS to the GPOS, and 2.64μs to the reverse situation. The measured memory footprint was about 1kbyte.

The research roadmap section described that research in the near future will focus on the development of a new shared device access mechanism, and the extension of the current architecture for multi-guest and multicore support. Research will then proceed towards the development of a trusted execution environment and its consolidation with the virtualization architecture.

VII. Acknowledgements

This work is supported by FEDER through COMPETE and national funds through FCT Foundation for Science and Technology in the framework of the project FCOMP-01-0124-FEDER-022674.

References

[1] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Computer Society*, vol. 38, no. May, pp. 39–47, 2005.

[2] S. Muthunagai, C. Karthic, and S. Sujatha, "Efficient access of Cloud Resources through virtualization techniques," *2012 International Conference on Recent Trends in Information Technology*, pp. 174–178, Apr. 2012.

[3] G. Heiser, "The role of virtualization in embedded systems," *Proceedings of the 1st workshop on Isolation and integration in embedded systems - IIES '08*, pp. 11–16, 2008.

[4] ——, "Virtualizing embedded systems-why bother?" *Proceedings of the 48th Design Automation Conference (DAC)*, pp. 901–905, 2011.

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

[6] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture," *Proceedings of the 5th European conference on Computer systems*, pp. 209–222, 2010.

[7] D. Rossier, "EmbeddedXEN: A Revisited Architecture of the XEN hypervisor to support ARM-based embedded virtualization," *White Paper*, 2012.

[8] A. Tavares, A. Didimo, T. Lobo, P. Cardoso, J. Cabral, and S. Montenegro, "RodosvisorAn ARINC 653 quasi-compliant hypervisor: CPU, memory and I/O virtualization," *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2012.

[9] VMware, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist," *White Paper*, pp. 1–14, 2007.

[10] P. Varanasi and G. Heiser, "Hardware-supported virtualization on ARM," *Proceedings of the Second Asia-Pacific Workshop on Systems*, 2011.

[11] A. Aguiar and C. Moratelli, "Hardware-assisted virtualization targeting MIPS-based SoCs," *23rd IEEE International Symposium on Rapid System Prototyping (RSP)*, pp. 2–8, 2012.

[12] ARM, "ARM Security Technology - Building a Secure System using TrustZone Technology," Tech. Rep., 2009.

[13] ——, "ARM Architecture Reference Manual - ARMv7-A and ARMv7-R edition," *DDI 0406C.b*, 2012.

[14] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," *ARM white paper*, vol. 3, no. 4, pp. 18–24, 2004.

[15] T. Frenzel, A. Lackorzynski, A. Warg, and H. Härtig, "ARM TrustZone as a Virtualization Technique in Embedded Systems," *Twelfth Real-Time Linux Workshop*, 2010.

[16] G. Platform, "The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market," *Global Platform white paper*, pp. 1–26, 2011.