Dual-OS Infrastructure for Mixed-Criticality Systems on ARMv8 Platforms

Alexander Spyridakis, Petar Lalov, Daniel Raho
Virtual Open Systems
Grenoble - France
Email: {a.spyridakis, p.lalov, s.raho}@virtualopensystems.com

***Abstract*—ARM devices are proliferating the mobile and embedded market segments, and with the introduction of Virtualization Extensions (ARMv7-A), and the latest 64-bit architecture changes (ARMv8), ARM is expected to expand further in the networking and server markets. At the same time, Mixed-Criticality use cases for In-vehicle (IVI) and In-flight (IFI) infotainment are of increased interest, where a feature rich Operating System (OS) is required for multimedia applications, while at the same time legacy real time operating systems are still needed for time critical applications. In this paper, we propose and test a Dual-OS environment for Mixed-Criticality systems using ARM devices, by exploiting latest architecture changes and software advancements. The technologies tested and covered in this paper, which enable a Dual-OS environment, include the TrustZone Security Extensions, ARMv7/v8 Virtualization Extensions, as well as the ARM Trusted Firmware (ATF) software infrastructure. Feasibility tests and latency/performance metrics were acquired on ARMv7/v8 platforms including Versatile Express and the Juno development boards.**

Keywords–Mixed-Criticality; ARM; embedded-virtualization; Dual-OS; real-time; Linux; KVM; GPOS and RTOS

I. INTRODUCTION

Real-time systems have pre-defined timing constraints and are deterministic in nature, thus a Real-Time Operating System (RTOS) has the ability to execute tasks with low latency, which are guaranteed to be completed on a predetermined deadline [1] [2]. On the other hand a General Purpose Operating System (GPOS), for example Linux, is targeting best performance instead of providing latency guarantees.

In the context of automotive, some subsystems are time critical, e.g., Electronic Stability Control (ESC) or Adaptive Cruise Control (ACC), while others are tied to multimedia services which are of low priority, with less to no criticality concerns. In such a Mixed-Criticality use case, there is no definite Operating System that can meet all needed characteristics, e.g., strict determinism, low latency, performance, portability, certifiability, feature richness, ease of development and maintenance.

Additionally, providing more performance by just increasing the clock frequency of the CPU is no longer feasible, instead the trend in current System on Chip (SoC) solutions, is to increase the number of cores and lower power dissipation. A continuous growth of multi-core platforms is being observed over the years, which essentially creates incentives for an efficient overcommitment of available resources and the combination of hardware for multiple purposes, which results in a lower total cost. For this reason, with the abundance of multi-core SoCs, it is no longer cost efficient to have multiple hardware instances with different software, instead the use of different Operating Systems in the same hardware platform is desired.

A. Contributions of this paper

First, we highlight the concept of a Dual-OS environment on modern ARM platforms and how a GPOS, such as Linux, can co-exist with other Operating Systems by using the TrustZone technology along with a novel firmware/monitor layer to handle interrupts, context switches and shared resources. Then, by leveraging Linux and KVM on ARM, we show how a host/guest OS can interface with an isolated RTOS running in the secure world. Additionally, we show how an efficient coordination scheduling mechanism can be implemented, to dynamically change scheduling policies triggered by synchronous and asynchronous events.

Finally, experimental results are reported, where the latency overhead of the ATF firmware layer is measured, as well as the communication overhead between a KVM guest and a TrustZone isolated bare-metal binary. The selected hardware platform for testing and benchmarking is ARM's latest 64-bit development board called Juno.

B. Organization of this paper

In Section II, we describe the overall architecture of a Dual-OS infrastructure on a modern ARM platform, and describe how the TrustZone security extensions can be combined with Virtualization Extensions to run two isolated Operating Systems with the option of also adding further Virtual Machines. Additionally, in Section III a list of previous related work is provided, along with how this paper contributes further to the concept. Then, in Section IV we document the ATF firmware layer that is responsible for handling the secure and non-secure world context switches, as well a basic description of its components. In Section V, we provide details on the ATF modifications needed for experimental measurements, along with actual results from the Juno and Versatile Express development boards. Finally, we conclude the work in this paper and list further possible directions.

II. DUAL-OS USE CASE IN IVI

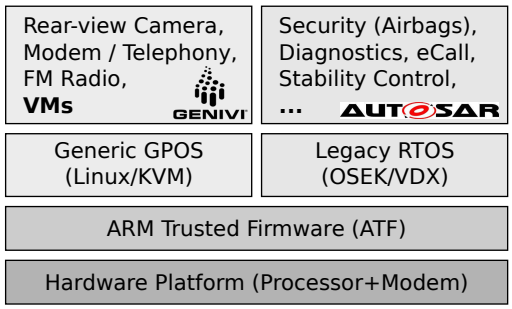
The target use case, similar to [3], is the deployment of two Operating Systems, one RTOS and a GPOS on a single multi-core hardware platform. In an automotive or even aerospace use-case, the medium of travel has an important placement

 ISBN: 978-1-61208-419-0

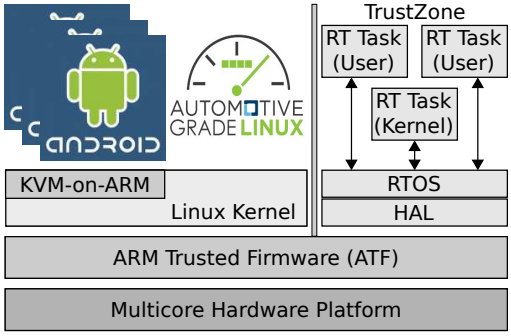
in the Internet of Things (IoT) arena. More specifically in the automotive example, high speed mobile communication enables the car to take the role of a gateway for connected objects.

Moreover, modern technology standards for modems such as 4G/4G+, enable the connection of multiple automotive devices. Applications that interface with the Long Term Evolution (LTE) software stack [5] will need to share resources both in the GPOS and RTOS, which will require a scheme of LTE virtualization either as direct device assignment or hardware assisted virtualization for the target device. Other protocols that are related to this use-case are the Controller Area Network (CAN) bus, as well as the high bandwidth communication IEEE 802.1 AVB Ethernet bus [4], which will also have to be accessible by the multiple actors (OSes) in the system.

Finally, similar to [6], this particular use case depends on the functionality of the GPOS to utilize the Virtualization Extensions of latest ARM architectures, for the instantiation of Virtual Machines completely isolated from the RTOS. This is possible with Linux and KVM on ARM, which allows the kernel to also act as a full-fledged hypervisor. Figure 1 shows the generic architecture of this use case.



(a) Automotive Dual-OS use case



(b) GPOS on KVM & RTOS in TrustZone

Figure 1. GPOS and RTOS on an Automotive Platform

For this mentioned automotive use case, the latest ARMv8-A architecture iteration, is targeting the full spectrum of high-end performance in embedded platforms, but at the same time keeping power needs to a minimum. The most significant change in this new architecture is the transition to 64-bit computing, while preserving 32-bit compatibility for legacy systems. As it was the case with its ARMv7-A predecessors, it provides Virtualization Extensions that enable its efficient usage for virtualization needs, as well as an isolated execution environment called TrustZone for secure computing. In general ARMv8-A brings the most advanced ARM features extended for 64-bit environments, which makes this architecture an ideal selection for virtualization and Dual-OS use cases.

A. Security Extensions

In modern CPU architectures, execution is split in multiple operational modes, with different security aspects and a fine-grained granularity to various instructions. The most obvious use-case for this paradigm is the separation of the kernel-space to user-space execution, where in user-space the permission rights for specific actions are reduced, while in kernel-space most instructions are available and the kernel has almost total control of the hardware.

In x86, this scheme is implemented with protection rings, where 4 different execution mode rings are available and the kernel/user -space code is placed in the most/least privileged mode respectively. For ARM devices these execution modes are called Supervisor and User mode, and newer architectures introduce additional modes such as the Secure Monitor and after-mentioned Hypervisor mode.

The ARM Security Extensions (a.k.a TrustZone) [7] [8], is a system-wide security approach for numerous client to server use-cases, including mobile devices, general purpose computers and enterprise systems. It can be utilized also as means to implement digital rights management, Bring Your Own Device scenarios and secure transactions. TrustZone is a core part of latest Cortex-A processors, although a complete implementation can be extended to the whole platform with specific TrustZone compatible devices/blocks, including secure memory, peripherals, accelerators, etc.

In essence, TrustZone adds a ”Secure” context to the available modes plus the addition of the Secure Monitor which is the most privileged CPU execution level. With this addition two instances of each mode (with the exception of Hypervisor mode which can only be non-Secure) can co-exist together, completely isolated from each other, where they are subject to the central authority of the Secure Monitor Exception Level 3 (EL) [9]. Practically, this means that with TrustZone you can have a Secure Supervisor or User mode (S-EL1/S-EL0) along with the previous non secure instances of these (EL1/EL0). With this set of features TrustZone allows the deployment of General Purpose Operating Systems such as Linux, together with Trusted Execution Environments (TEE). The secure modes have the same features as the normal ones, while operating in an isolated memory space. Finally, the Secure Monitor has utmost authority of all modes handling the world switch (context) between them.

B. Virtualization Extensions

With ARM attempting to break into new markets, but also trying to keep its dominance in existing segments, since ARMv7-A (Cortex-A15, A7, etc.) they have added a number of new features in the ARM architecture in order to facilitate virtualization, usually referred to as the ARM Virtualization Extensions [10].

A new processor mode is introduced, called Hypervisor mode, which allows each guest to have access to its own privileged mode; the processor’s state can be switched between

 ISBN: 978-1-61208-419-0

guests, allowing the processor to be virtualized without expensive binary patching techniques, and with very few traps being necessary. The Virtualization Extensions also allow certain instructions to be set up to trap to the hypervisor if that is necessary to support certain guest features.

The ARM Virtualization Extensions also include functionality to assist with the virtualization of memory for guests. For this, the Large Physical Address Extensions [11], besides an updated page table format also include the possibility to set up a second stage of memory translation to be used by the hypervisor.

The above described extensions are sufficient to fully virtualize the CPU and memory for any number of guests, and also trap any accesses I/O devices so they can be emulated by software. These satisfy the requirements to implement an efficient native virtualization solution on newer ARM processors.

III. Related Work

The concept of Dual-OS in embedded systems has been explored previously, dealing mostly with ARM devices by leveraging TrustZone. Yoshinori Endo et al [18], propose a generic architecture of dual operating systems in automotive called Dependable Autonomous hard Real-time Management (DARMA), with an implementation based on an SH-4 RISC processor.

For ARM platforms, Daniel Sangorrin et al [20], give details on a thin Secure Monitor layer called SafeG and provide examples of scheduling and device sharing between the two Operating Systems on an ARMv6 platform. Finally, Soo-Cheol Oh et al [19], implement their own solution called ViMoExpress based on a Cortex-A8 processor and an LCD virtualization feature.

For this paper, the added contribution is the application of the concept on latest ARMv8 platforms, with the possibility to run 64-bit or legacy 32-bit software, as well as the combination of multiple guest operating systems by utilizing KVM and the virtualization extensions of the ARM architecture.

IV. ARM Trusted Firmware

For a Dual-OS environment we need a firmware layer that will make use of the TrustZone security extensions to isolate resources to their secure and non-secure equivalents. For our firmware needs, which also fit the Juno hardware platform, ARM Trusted Firmware is selected.

ATF [12] is a secure world software implementation for ARMv8-A platforms, provided as a reference firmware infrastructure. Additionally, ATF is meant to be a modular framework for handling the boot procedure, interrupt management and world switching on all available SoC cores. Modularity is key for portability and maintainability, as well as covering any separation concerns in the firmware level, resulting in easier development/testing and certification. At its current state ATF is a standardized EL3 Runtime Firmware for all 64-bit ARMv8-A platforms, with plans to extend it further to cover older ARMv7-A architectures. Although ATF is designed as a firmware solution to run multiple OSes in parallel, it can still be used in cases where only one operating system is used, without a strict requirement of a Secure-OS. The top-level architecture of ATF can be summarized in Figure 2, the functionality overview is also listed below:

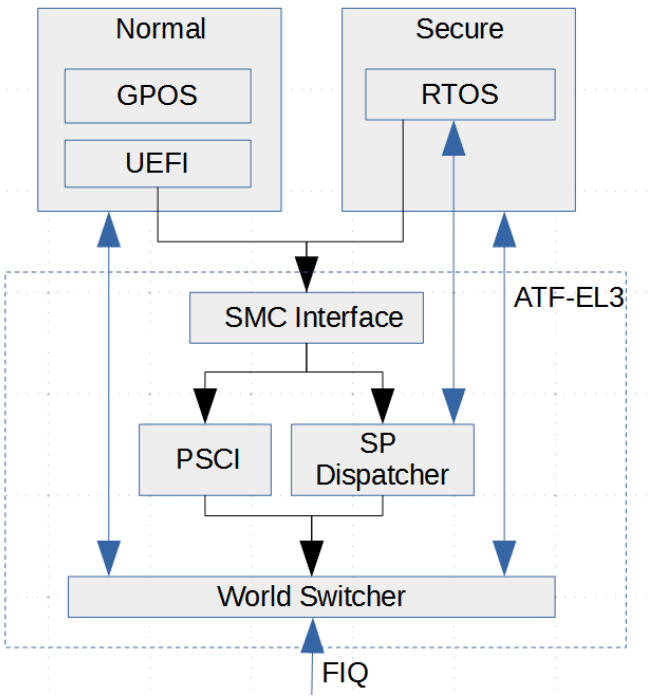


Figure 2. ATF architecture

- Secure world Initialization (e.g., exception vectors, interrupt handling, registers, etc.).
- Support for the newer Generic Interrupt Controllers found in latest ARM devices, which are also virtualization aware and compatible with TrustZone.
- Proper initialization of the Normal world, typically in AArch64 EL2 mode, which is also required for KVM initialization by the kernel.
- Handles Secure Monitor Call (SMC) requests from booted Operating Systems for PSCI power management features such as, booting secondary cores, hotplug and shutdown/reset events.
- Secure-EL1 Payload Dispatcher for handling world switching and interrupt routing.
- Option to replace the Trusted Boot Firmware adapted for the needs of the target platform.
- Memory isolation from secure/normal world based on the features provided by TrustZone.

Figure 3 depicts an overview of the boot procedure in ATF, with the execution sequence between the blocks/modules that ATF consists of. Every stage of the ATF Boot Loader has a dedicated purpose during initialization system boot and any of the following listed BLs can be replaced by custom implementations according to the target platform needs and requirements.

 ISBN: 978-1-61208-419-0

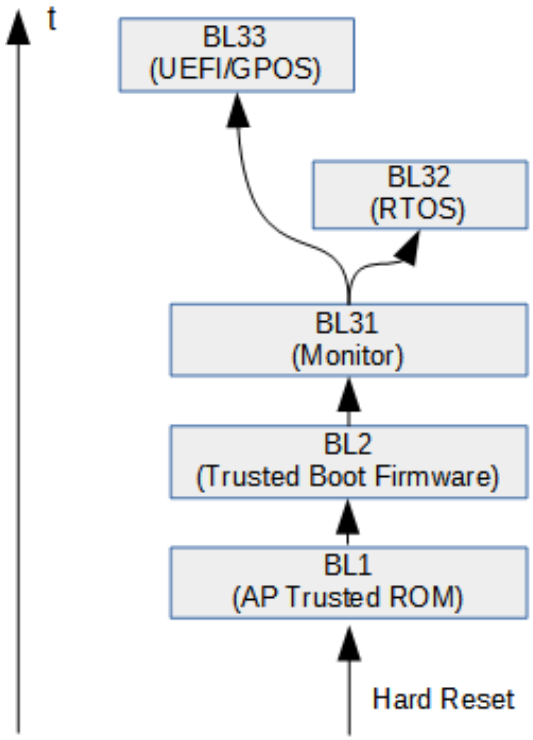


Figure 3. Top-level ATF execution flow

- BL1 - AP Trusted ROM - basic initial boot up procedure in EL3, which subsequently calls BL2.
- BL2 - Trusted Boot Firmware - responsible to pass execution to further BL3-x modules.
- BL3-1 - EL3 Runtime Firmware - Exception routing to dedicated handlers in the corresponding world (world-switch).
- BL3-2 - Secure-EL1 Payload (optional) - runs in SEL-1 and could be an RTOS or another secure bare-metal binary
- BL3-3 - Non-trusted Firmware - any non-secure firmware/software, e.g., u-boot, UEFI (Tianocore EDK2), GPOS, etc.

A. Interrupt management & world switching

In general, exception routing and world switching is implemented at the BL3-1 level, and since an actual Interrupt Service Routine is not part of the EL3 core logic, instead, for the final handling of interrupts, ATF defines interfaces for the retransmission of exceptions to the end destination OS. More specifically, FIQ routing, is managed by the Secure-EL1 Dispatcher layer, while SMC exceptions are standardized in the EL3 runtime service framework based on the SMC Calling Convention PDD. Finally, the EL3 runtime service interface, is complemented by the Power State Coordination Interface (PSCI).

Secure-EL1 Dispatcher service, as part of the EL3 runtime service, is a link interface between BL3-2 (Secure OS/Payload) and BL3-1 (ATF). It is responsible for processing the entry/exit requests for the target secure software (e.g., RTOS), and is designed in a way to provide a common calling convention between these two layers. This service is implemented according to the needs/requirements of the deployed secure software.

PSCI [13] is responsible for the power management of all available SoC cores, where it also supersedes the old mechanism of waking up secondary cores, also known as the CPU holding pen. With PSCI an Operating System can signal the firmware layer to power up/down available cores, through the use of SMC/Hypervisor Call (HVC) instructions. This new paradigm considerably simplifies power management for an OS, as instead of having to implement low level target specific power routines, the OS can rely on the firmware layer.

B. ATF modifications for Dual-OS

By default ATF is using time-triggered signals to change between the secure/non-secure payloads. For this purpose, the internal architected ARM timer [14] is programmed to fire as a secure interrupt and signal a world switch, this interrupt is configured as a Fast Interrupt Request (FIQ) and is handled according to the current context. If at the time of the received FIQ the non-secure world (GPOS) is active, execution will be immediately directed to EL3 for a world switch to the secure payload (RTOS). On the other hand if the secure payload has the context, the interrupt is by default serviced by itself, without the need of EL3 interception.

For experimental measurements, we opted for an event-triggered implementation, where the world switch is triggered by the SMC instruction. That way each world can yield CPU resources deterministically, and give back the context when this is desired. Modifications on the Secure-EL1 Dispatcher service were needed for this behavioral change, which allows us to have a more fine-grained control for latency measurements.

V. Experimental results

Experimental results are split in two sections. First we deploy ATF with two separate world switch triggering methods, and we measure the overall latency introduced when switching between the Secure and non-Secure worlds. Subsequently, QEMU is used to create a KVM accelerated virtual machine and measure the guest exit overhead, that results when an SMC or HVC instruction is called from the guest.

By using the Performance Monitor Unit (PMU), which is found on all recent ARM CPUs, we can precisely measure programmatically the latency between two points in execution. The PMU [15] provides a number of counters and registers for gathering statistics on the operations executed in the processor. Among a set of configurable event and performance counters the PMU provides a 64-bit cycle counter, which is incremented on every clock cycle. For the following measurements the PMU cycle counter (PMCNT) is reset to zero before starting the process of a world/context switch and is stopped after execution is resumed in the respective world.

A. ATF world switch latency

As described in Section IV-B, ATF by default is using time-triggered signals to issue a world switch. For a more deterministic approach ATF was modified so that a world

 ISBN: 978-1-61208-419-0

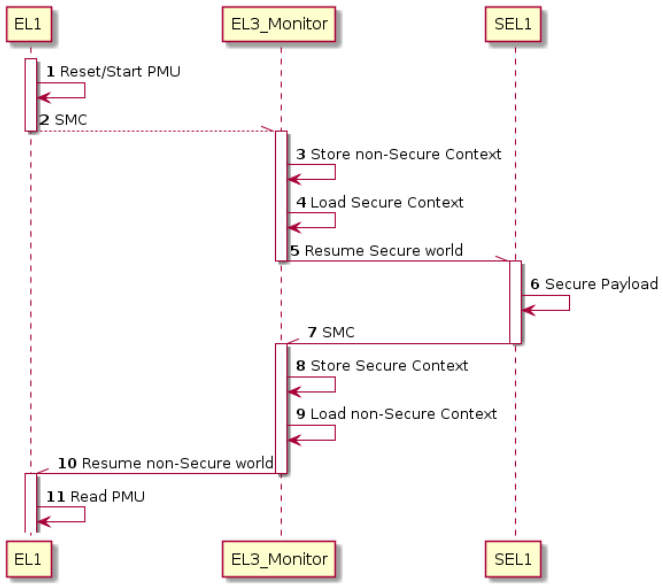


Figure 4. Execution flow of world switch measurement

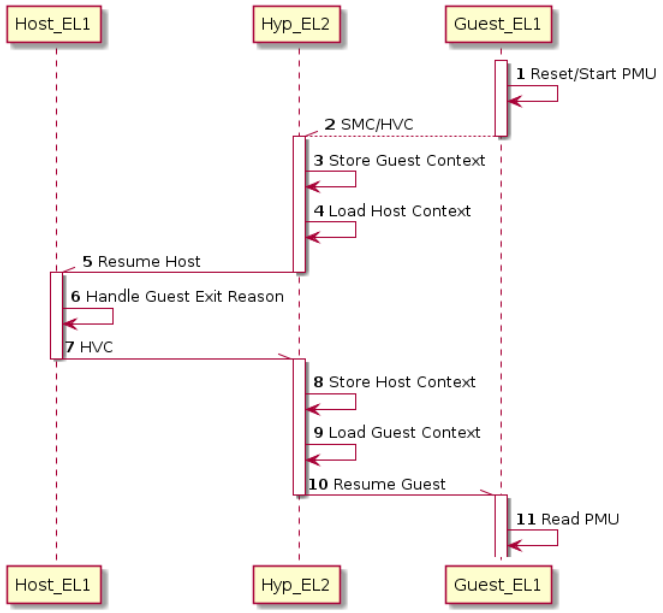


Figure 5. Execution flow of guest/host context switch measurement

TABLE I. LATENCY FOR A FULL WORLD-SWITCH IN CLOCK CYCLES

Average	Minimum	Maximum	Std. deviation
3716,66	3520	3998	95,245

switch can occur programmatically when issuing the SMC instruction.

In the case of Linux, we implemented a small module which at the time of its loading it would reset the PMU cycle counter and immediately issue an SMC instruction. This Secure Monitor Call is then trapped by ATF, which saves the non-Secure context (Linux) and then proceeds to restore and resume the Secure payload. On the Secure side, the SMC is re-issued immediately to trap back to ATF and finally resume execution in Linux.

Figure 4 depicts how latency was acquired in relation to the execution flow. Measurements were repeated 1000 times and results are reported in Table I. The reported results show that a full world switch (non-Secure to Secure and back) is in the range of just under 4000 cycles. This eventually means that the Secure payload, at any point of execution, will receive control of the system in less than 2000 cycles, with actual values in time depending on the clock frequency of the processor.

B. Virtual machine context switch latency

For guest to host latency measurements a similar approach to the world switch measurement is made. The difference is that for the guest we use a thin bare-metal program instead of a full Linux guest. The guest executes a loop where the PMU cycle counter is reset and immediately an SMC or HVC instruction is issued. After guest execution is resumed the PMU cycle counter register is saved to memory and the process is repeated for 1000 times. Before the guest is terminated results are reported to the user.

From the host side, KVM will not allow the guest to execute an SMC or HVC instruction, as they are considered ”sensitive” instructions and are immediately trapped by the hypervisor. An example of this interaction is how KVM wakes up secondary guest cores through the use of PSCI, in which the guest will call HVC with the proper argument. KVM traps the guest, checks the provided argument and decides if the HVC instruction was meant to be a PSCI wake up event. In our case, as PSCI is not used, KVM will inject an abort exception and the guest will be halted. For this reason KVM needs to be modified in order to resume execution to the guest without aborting. A similar usage pattern of HVC instructions, has already been highlighted as means to implement a Storage I/O co-ordination scheduling approach, between a Linux/KVM host and a guest system, with significant improvements in latency and responsiveness of guest applications [16] [17].

Once again Figure 5 highlights the execution flow of the measurement and results are reported in Table II. It is interesting to note that this measurement was replicated in both an ARMv7-A target (Versatile Express TC1 - Cortex-A15) and an ARMv8-A platform (Juno board - Cortex-A53). Furthermore the ARMv8-A Virtualization Extensions with KVM provide a way to run legacy ARMv7 guests, but results were not affected by this scenario. Finally, results show a full host/guest context switch (guest to host and back) of around 1500 cycles for both ARMv7 and ARMv8 platforms. Coupled with the world switch latency results, it means that if a guest needs to be interfaced and communicate with the Secure software, a latency of at least 5500 - 6000 cycles is expected on this firmware implementation.

Summarizing the results, with a reference CPU clock of 1GHz the average latency for a full world switch (non-Secure to Secure and back) is in the range of 4 μs. A similar path for a guest to host context switch (and back to guest) is around 1,5 μs.

 ISBN: 978-1-61208-419-0

TABLE II. LATENCY FOR A FULL GUEST CONTEXT-SWITCH IN CLOCK CYCLES

Type	Average	Minimum	Maximum	Std. deviation
SMC - ARMv8	1497,88	1475	1906	27,766
HVC - ARMv8	1492,62	1461	1932	39,697
SMC - ARMv7	1647,58	1612	2429	66,130
HVC - ARMv7	1679,46	1655	3115	50,140

VI. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we proposed the use of a Dual-OS infrastructure to efficiently leverage ARM multi-core hardware platforms in Mixed-Criticality use cases. Through architecture technologies such as TrustZone and a proper firmware layer like ATF, two completely independent and isolated OS instances can be run simultaneously on the same hardware resources. Additionally, by using Linux as the GPOS and KVM on ARM, we can extend the number of concurrent Operating Systems even further.

Experimentally, with the ARMv8 Juno development platform, we highlighted the latency overhead of a secure/non-secure world context switch and interrupts handled by ATF, as well as the equivalent latency when switching from a guest OS to the host with KVM (including measurements with ARMv7 Versatile Express). Such measurements show the feasibility of the Dual-OS concept, with an average world switch latency of around 4 μs and guest to host switch of 1,5 μs on our reference platforms.

Future work, will include a communication interface between GPOS/RTOS and VMs, in order to implement a complete TEE solution and a fine-grained co-scheduling policy in the system. Finally, the ATF firmware layer is going to be replaced by a fully redesigned custom bare-metal software that will be targeting automotive certification.

ACKNOWLEDGMENTS

This research work has been supported by the Seventh-Framework Programme (FP7/2007-2013) of the European Community under the grant agreement no. 610640 for the DREAMS project.

REFERENCES

[1] J. Altenberg, “Using the Realtime Preemption Patch on ARM CPUs,“ 11th Real-Time Linux Workshop (RTLW 09), Sep 2009, pp. 229-236.

[2] K. Koolwal, “Myths and Realities of Real-Time Linux Software Systems,“ 11th Real-Time Linux Workshop (RTLW 09), Sep 2009, pp 13-18.

[3] M. Hamayun, A. Spyridakis, and D. Raho, “Towards Hard Real-Time Control and Infotainment Applications in Automotive Platforms,“ 10th annual workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT 2014), July 2014, pp 39-44.

[4] R. Kreifeldt, “AVnu Alliance White Paper: AVB for Automotive Use,“ [retrieved: June 2015]. [Online]. Available: http://www.avnu.org.

[5] “Long Term Evolution Protocol Overview,“ [retrieved: June 2015]. [Online]. Available: https://rt.wiki.kernel.org/index.php/Cyclictest

[6] H. Mitake, Y. Kinebuchi, A. Courbot, and T. Nakajima, “Coexisting Real- time OS and General Purpose OS on an Embedded Virtualization Layer for a Multicore Processor,“ Proceedings of the 2011 ACM Symposium on Applied Computing (SAC 11), 2011, pp. 629-630.

[7] “ARM TrustZone,“ [retrieved: June 2015]. [Online]. Available: http://www.arm.com/products/processors/technologies/trustzone/index.php

[8] “ARM Security Technology,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[9] “ARM Exception Levels,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0438d/CHDCGIBF.html

[10] “ARM Virtualization Extensions,“ [retrieved: June 2015]. [Online]. Available: http://www.arm.com/products/processors/technologies/virtualization-extensions.php

[11] “ARM LPAE Architecture,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0438d/CHDCGIBF.html

[12] “ARM Trusted Firmware,“ [retrieved: June 2015]. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware

[13] “Power State Coordination Interface,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.den0022c/DEN0022C_Power_State_Coordination_Interface.pdf

[14] “Generic Timer architecture,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0438d/BABBJGCH.html

[15] “Performance Monitor Unit,“ [retrieved: June 2015]. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0363e/CHDCBAAH.html

[16] A. Spyridakis, and D. Raho, “On Application Responsiveness and Storage Latency in Virtualized Environments,” CLOUD COMPUTING 2014, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp. 26-30.

[17] A. Spyridakis, D. Raho, and J. Fanguède, “Virtual-BFQ: A Coordinated Scheduler to Minimize Storage Latency and Improve Application Responsiveness in Virtualized Systems,” International Journal on Advances in Software, vol 7 no 3 & 4, 2014, pp. 642-652.

[18] Endo, Yoshinori, et al, “In-Vehicle Multimedia Platform Based on Darma (Dual Os Approach),” Proc. 7th World Congress on Intelligent Systems, Turin, Italy. 2000.

[19] O. Soo-Cheol, K. Koh, C. Kim, K. Kim, and SeongWoon Kim, “Acceleration of dual OS virtualization in embedded systems,” In Computing and Convergence Technology (ICCCT), 2012 7th International Conference on, pp. 1098-1101. IEEE, 2012.

[20] D. Sangorrin, S. Honda and H. Takada, “Reliable device sharing mechanisms for Dual-OS embedded trusted computing,” TRUST’12 Proceedings of the 5th international conference on Trust and Trustworthy Computing, pp. 74-91.

 ISBN: 978-1-61208-419-0