A TrustZone-assisted Hypervisor Supporting Dynamic Partial Reconfiguration

José Ribeiro
Centro Algoritmi
Universidade do Minho
a71881@alunos.uminho.pt

Nuno Silva
Centro Algoritmi
Universidade do Minho
a70616@alunos.uminho.pt

Sandro Pinto
Centro Algoritmi
Universidade do Minho
sandro.pinto@dei.uminho.pt

Adriano Tavares
Centro Algoritmi
Universidade do Minho
atavares@dei.uminho.pt

***Abstract*—Reconfigurable systems have been proven to be very powerful in terms of flexibility and performance. With virtualization being imperative in many modern embedded systems, the juxtaposition between this two key technologies was imminent. Hypervisors like the Mini-NOVA enforce this but neglect fundamental features, such as not making use of hardware extensions to help with virtualization and not taking full advantage on the offloadable features to reconfigurable hardware modules.**

This work in progress paper presents a Microkernel-like TrustZone-assisted Real-Time Hypervisor that dynamically manages software and reconfigurable hardware tasks, combining these technologies with main focus on Dynamic Partial Reconfiguration (DPR) towards a promising solution, as shown by the encouraging preliminary results. It is also discussed our research roadmap for the future.

Keywords*—*Virtualization, ARM TrustZone, Hypervisor, FPGA, Dynamic Partial Reconfiguration

I. Introduction

The growing necessity of heterogeneous environments around embedded system solutions has led to a rise in the complexity of these systems. To manage such increase in complexity while keeping under control metrics such as size, weight, power and cost (SWaP-C), virtualization has become a normalised practice. To further complement this approach, and given the great emphasis to flexibility and performance on some safety-critical environments, there is a high investment in exploring Field-Programmable Gate Arrays (FPGA) assisted approaches [1].

Virtualization technology allows both general-purpose computing and real-time requirements to be achieved by enabling concurrent execution of multiple Virtual Machines (VMs) on the same hardware processor. Unfortunately, these virtualized embedded systems may not be able to meet the real-time demands aforementioned, as a result of the performance overhead induced to the processor cores by the traditional software-based technology. To solve this issue, many hardware extensions (i.e. Intel's Virtualization Technology (VT) [2], [3], Arm's Virtualization Extensions (VE) [4], [5] and TrustZone (TZ) [6], [7]), have been developed to support these systems.

Towards achieving better solutions for higher-performance virtualized embedded systems, a new architecture based on the combination of processor cores with hardware extensions and FPGA fabric emerged [8]. This design strategy has been scrutinised for many years from a myriad of different angles, whether by only making use of the virtualization hardware extensions [9], [5], [10], [11], or by also offloading software functionalities to FPGA components [12]. However, more recently, the use of DPR has been investigated on a wider range of embedded systems' domains, but not constrained to virtualized systems. This is evident in [13], where DPR technology is used as a powerful flexibility tool for an industrial system and, in the embedded systems' domain, DPR hardware modules have also been shown to work concurrently with real-time software tasks on a Real-Time Operating System (RTOS) [14]. Regarding virtualized systems, Xia's work on the Mini-NOVA Hypervisor [8], [15] has become a staple for DPR in virtualized embedded systems, making use of DPR alongside a traditional para-virtualized Hypervisor.

The goal of this work in progress is to create a Microkernel-like TrustZone-assisted Real-Time Hypervisor that dynamically manages software and DPR hardware tasks. This Hypervisor will present some key differences from the Mini-NOVA, namely: (1) fully-virtualized Hypervisor making use of TrustZone hardware extension, contrary to the para-virtualized nature of the Mini-NOVA; (2) offloading Hypervisor services, such as Inter-Partition Communication (IPC) mechanism with security capabilities, to DPR hardware peripherals, differently from the Mini-Nova, which only offloads RTOS tasks to DPR; (3) extend TrustZone capabilities to the DPR hardware peripherals making them secure and, as such, guaranteeing the hardware Trusted Computing Base's (TCB) integrity. The board chosen for this research work is the Xilinx's Zybo from the Zynq-7000 System-on-Chip (SoC) family [16], [17], as it contains all the components (TrustZone-enabled processor and peripherals as well as FPGA fabric) required for building the proposed prototype.

II. Background

A. ARM TrustZone

TrustZone technology consists on a set of hardware-based security extensions to ARM SoCs implemented since the ARMv6 architecture. These hardware security extensions provide a secure and separate execution environment that protects the integrity and confidentiality of secure-sensitive processing, by splitting the hardware and software resources into two worlds - the secure world and the non-secure world [6], [18].

The TrustZone hardware architecture can be seen as a dual-virtual system, which splits all the system's physical resources into two possible virtual environments. The major changes introduced in the hardware architecture include the ability to tag system resources as belonging to the secure or normal world. To indicate in which world the processor is executing, there is the new 33rd processor bit - *NS* (Non-Secure) bit, which is also extended to the rest of devices, enhancing control for the system designer over peripheral buses and memory [18], [19]. To preserve the processor state during the world switch, TrustZone adds an extra processor mode: the monitor mode. When running in monitor mode, the processor state is always considered secure. Since the processor only runs in one world at a time, software stacks in both worlds can be bridged via a new privileged instruction - Secure Monitor Calls (SMC). The monitor mode can also be entered by configuring it to handle interrupts and exceptions in the secure side.

The memory infrastructure outside the core can also be partitioned into the two worlds through the TrustZone Address Space Controller (TZASC). Dynamic Random Access Memory (DRAM) can be partitioned into distinct memory regions, each of which can be configured to be used in either world or both. The processor also provides two virtual Memory Management Units (MMUs), and isolation is still available at the cache-level. System peripherals can be also configured as secure or non-secure through the TrustZone Protection Controller (TZPC).

B. μRTZVisor

μRTZVisor [10] stands for Microkernel real-time TrustZone-assisted Hypervisor, and is an extended version of RTZVisor [11] for a Microkernel-like architecture, while following an object-oriented approach. μRTZVisor targets security not only from the outset but also from the onset [10], by applying a secure development process.

Contrarily to existing Microkernel-based solutions, μRTZVisor is able to run nearly unmodified guest Operating Systems (OSes), while also providing a high degree of functionality, configurability and real-time support. It implements a scheduling policy based on time domains, which can have different priorities and are scheduled according to a preemptive, round-robin schema. Performed experiments demonstrate a performance overhead around 2% for a 10 milliseconds guest time domains [10].

Also crucial on the μRTZVisor is its capability-driven Inter-Partition Communication (IPC) mechanism that works in conjunction with the real-time scheduler and memory subsystem (Figure 1) [10]. Such capability-driven IPC is dictated by the Microkernel-like architecture of the μRTZVisor in order to provide secure and efficient ways of sharing data among services, drivers and Guests.

III. The Proposed DPR-assisted μRTZVisor

The design and implementation of the μRTZVisor was tailored for a Zynq-7000 SoC [10] and so, the proposed architecture is divided into its two main functional blocks:

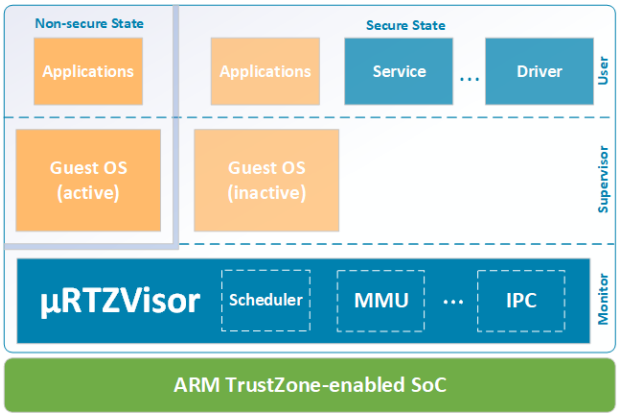

Fig. 1. μRTZVisor architecture.

the Processing System (PS) and the Programmable Logic (PL). The PS is the Central Processing Unit (CPU) and includes the software computing resources, such as the ARM Cortex-A9 processor, the On Chip Memory (OCM) and various peripherals. On the CPU, the μRTZVisor hosts guest applications/software and is responsible for scheduling these components properly. The PL will contain different hardware accelerators and execute concurrently with the PS side. The subsystems in the PS are interconnected among themselves, and connected to the PL side, through an ARM AMBA AXI Interconnect.

The SoCs of the Zynq family support DPR either under the control of the software running on the PS through the Device Configuration Interface (DevCfg), which launches a Direct Memory Access (DMA) transfer via the Processor Configuration Access Port (PCAP) or under the control of the hardware via the Internal Configuration Access Port (ICAP), which is capable of self-configuration from the PL side [20]. The major disadvantages of using ICAP, is the fact that it uses an AXI4-Lite port as a transfer port as well as requiring extra hardware resources. The use of Partial Reconfiguration (PR) allows the designer to define multiple Partial Reconfigurable Regions (PRR) in the FPGA and dynamically reconfigure them with different functionalities during runtime. Partial bitstreams are loaded to the PRRs via the previously mentioned ports. DPR also allows the system's power consumption to be reduced when compared to full reconfigurations each time a functionality change is required. Partial bitstreams are also smaller than full bitstreams and the reconfiguration time is shorter, which leads to less processing time. Other feature enabled by DPR, is that it can be used to reconfigure PRRs with blank bitstreams while they are inactive and not necessary, as long as the throughput for the configuration is high enough [21], reducing power consumption.

The proposed architecture is shown in Figure 2. μRTZVisor will be extended with a DPR Manager in order to dynamically support modifications/changes of specific parts of hardware components. This way, the FPGA resources can be considered as standard user applications making the simultaneous man-

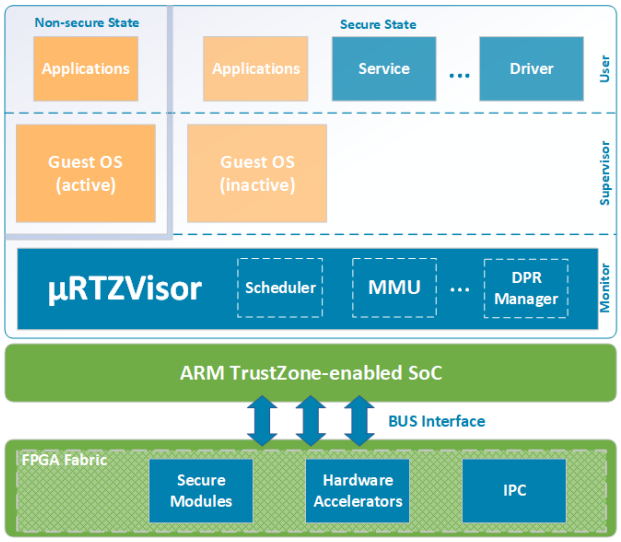

Fig. 2. DPR-assisted μRTZVisor architecture.

agement of hardware and software tasks possible. Therefore, the manager will allow the Hypervisor to reconfigure multiple PRRs which will focus on three main parts:

- Hardware Tasks: As DPR allows to reprogramme a part of the FPGA while the rest of the system is still running without interference, it is possible to introduce hardware tasks, which are tasks from the OSes that are offloaded to hardware. A set of PRRs will be used as containers for hardware tasks accelerators which are predefined bitstream files that hold the module fabric information for the desired functionality and any guest may require to use. When a guest sends the request to use a hardware task to the manager, it will check if the task is available for usage and if so, assign it to the guest. Different hardware tasks are dispatched by programming the assigned PRRs with the bitstream file for the specific task.
- Hardware IPC: Offload IPC to different hardware implementations, while testing the benefits of creating a dedicated IPC peripheral capable of accelerating data transfers and increasing security. This encompasses leveraging a capability-based access-control facility, which will enforce Information Control-Flow (ICF) and containment by communication relationships. Also, make use of secure DMA transfers to upgrade transmission rates while maintaining TrustZone features and coherency with the μRTZVisor and the existent IPC mechanism. Some of the hardware module components will be PR to benefit the desired data transfer.
- Security Modules: Expand the security spectrum of the Hypervisor by implementing security by diversity. This means implementing the same behaviour/functionality using different implementations. For instance, when the system is detected to be under attack, the DPR manager can reconfigure security modules through DPR to different implementations with similar behaviour.

IV. Preliminary Results

In order to better understand the impact of partial reconfiguration, a test was executed to measure reconfiguration times for different bitstream sizes. The results presented for this test have been collected in a test application in which the partial bitstream files were stored into the DDR3 memory.

The tests showed that each bitstream file describing the same PRR on the FPGA has exactly the same dimension and takes about the same time to be loaded to the PRR. Table I shows that on the Xilinx Zybo, with PCAP/DMA-based approach, the PCAP throughput is substantially constant and makes the reconfiguration time linearly proportional to the size of the partial bitstream files.

TABLE I
Reconfiguration Times

Bitstream Size (kB)	Avg. Reconfig. Time (ms)	Throughput (MB/s)
108	0.87	124.36
584	4.68	124.46
1282	10.28	124.67

Some Hypervisor benchmarks were executed on this early stage, with the intention of a later comparison with the modified system. The time of the context-switching operation for each scenario, running partitions in separate 10 ms time domains, was measured 10,000,000 times for each switching and the Worst Case Execution Times (WCET) are shown in Table II. These results were taken from [10].

TABLE II
Context Switch WCET (μs)

Guest–Guest	166.68
Task–Task	10.38
Guest–Task	19.13
Task–Guest	19.63
Task–Different Guest	156.00

Also in [10], the asynchronous IPC performance was measured but only reflecting the time that it takes to perform the respective hypercalls from a guest partition. For a 64 byte message size, the hypercall execution time is of 4.36, 4.17 and 5.49 μs for each operation, increasing by about 1 μs for each additional 64 bytes in the message, as can be seen in Table III.

TABLE III
Asynchronous IPC primitives latency (μs)

Message Size (bytes)	Send	Receive	Send Receive
64	4.36	4.17	5.49
128	5.17	4.75	6.31
192	6.00	5.16	7.13
256	6.82	5.72	7.98
320	7.69	6.21	8.8

The synchronous IPC performance, which follows the typical client-server communication scenario, was also evaluated on a Guest-to-Guest, Guest-to-Task and Task-to-Task scheme. in Table IV, the Guest-to-Guest times are shown.

TABLE IV
SYNCHRONOUS GUEST-TO-GUEST IPC COMMUNICATION LATENCY (μS)

Message Size (bytes)	Send	One-Way	Two-Way
64	15.21	195.14	385.73
128	16.24	197.23	389.45
192	16.78	199.76	394.18
256	18.58	202.65	398.10
320	18.88	204.66	402.39

V. RESEARCH ROADMAP

So far, the controller is capable of reconfiguring PRRs dynamically with functional hardware accelerators. However, it is not yet integrated with the Hypervisor. As such, the next essential step is to integrate the DPR software manager on the Hypervisor to trigger the hardware reconfiguration. To synchronise the DPR hardware modules with the software manager, a fixed hardware module will be implemented which will hold information about the modules.

Following is the implementation of the hardware modules for each of the three intended FPGA-based features, as shown in Fig. 2. Then, our focus will be centred on making all the PRRs coexist together, which will require the distribution of the FPGA resources according to the modules' needs. Once all the modules are implemented and the wanted controller behaviour is achieved, in such a way that it can control all the PRRs dynamically, we will work on refining the DPR hardware controller to guarantee the validation of the information it holds.

Upon reaching this stage, we want to exhaustively evaluate the controllers in the performance, memory footprint, hardware cost and energy consumption metrics. The research will then continue towards improvements in the controller performance in order to optimise hardware usage and tasks management.

VI. CONCLUSION

Minding the great emphasis given to DPR technology on recent research, it has become one of the key technologies in the embedded systems field. Given its potential for deploying hardware modules on-the-fly, DPR has the ability to bring immense flexibility to a system, as well as an overall performance improvement. The merging of this technology with virtualization is becoming eminent, considering the benefits of carrying DPR to the virtualized embedded systems' domain, as proven by Xia's Mini-NOVA implementation. However, there are some oversights with this implementation.

This paper presents a work in progress on the implementation of a Microkernel-like TrustZone-assisted Real-Time Hypervisor that dynamically manages software and DPR hardware tasks, while only introducing a low performance cost from the reconfiguration process and adding the physical cost of the hardware components.

The research roadmap section described the work for the near future, which will focus on the development of the DPR controller and the hardware modules, while also integrating with the chosen Hypervisor.

VII. ACKNOWLEDGMENTS

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT - *Fundação para a Ciência e Tecnologia* within the Project Scope: UID/CEC/00319/2013.

REFERENCES

[1] M. Valdes Pena, J. Rodriguez-Andina and M. Manic, "The Internet of Things: The Role of Reconfigurable Platforms," *IEEE Industrial Electronics Magazine*, vol. 11, no. 3, pp. 6–19, 2017.

[2] G. Neiger, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization," *Intel Technology Journal*, vol. 10, no. 3, pp. 167–178, 2006.

[3] R. Uhlig, et al., "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[4] R. Mijat and A. Nightingale, "Virtualization is coming to a platform near you," *ARM White Paper*, pp. 1–12, 2011.

[5] T. Shimada, T. Yashiro, N. Koshizuka, and K. Sakamura, "A real-time hypervisor for embedded systems with hardware virtualization support," *in Proc. 2O15 TRON Symposium (TRONSHOW)*, 2016.

[6] ARM Limited, "ARM Security Technology - Building a Secure System using TrustZone Technology," *ARM White Paper*, 2009.

[7] A. M. Azab, et al., "Hypervision Across Worlds : Real-time Kernel Protection from the ARM TrustZone Secure World," *in Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[8] T. Xia, J. C. Prevotet, and F. Nouvel, "Mini-NOVA: A Lightweight ARM-based Virtualization Microkernel Supporting Dynamic Partial Reconfiguration," *in Proc. 2015 IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 71–80, May 2015.

[9] S. Pinto et al., "Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone," *in Proc. 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014.

[10] J. Martins, J. Alves, J. Cabral, A. Tavares, and S. Pinto, "μRTZVisor: a Secure and Safe Real-Time Hypervisor," *Electronics*, vol. 6, no. 4, 2017.

[11] S. Pinto, A. Tavares, and S. Montenegro, "Hypervisor for Real Time Space Applications," *in Proc. The 4S Symposium*, 2016.

[12] P. Garcia, et al., "On-chip message passing sub-system for embedded inter-domain communication," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 33–36, 2016.

[13] R. Wisniewski, G. Bazydlo, L. Gomes, and A. Costa, "Dynamic Partial Reconfiguration of Concurrent Control Systems Implemented in FPGA Devicesvai por," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1734-1741, 2017.

[14] M. Pagani, et al., "Towards Real-Time Operating Systems for Heterogeneous Reconfigurable Platforms," *OSPERT 2016*, September 2016.

[15] T. Xia, J. Prévotet, and F. Nouvel, "An ARM-based Microkernel on Reconfigurable Zynq-7000 Platform," *Mediterranean Telecommunication Journal*, vol. 5, no. 2, pp. 109–115, 2015.

[16] Xilinx (Oct. 20, 2017), Zynq-7000 All Programmable SoC (v1.12). [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. Accessed on: Nov. 28, 2017.

[17] Xilinx (May 6, 2014), Programming ARM TrustZone Architecture on the Xilinx Zynq-7000 All Programmable SoC (v1.0). [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1019-zynq-trustzone.pdf. Accessed on: Nov. 28, 2017.

[18] J. Winter, "Trusted computing building blocks for embedded linux-based ARM trustzone platforms," *in Proc. 3rd ACM workshop on Scalable trusted computing*, pp. 21-30, 2008.

[19] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM trustzone to build a trusted language runtime for mobile applications," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 67-80, 2014.

[20] Xilinx (Apr. 5, 2017), Vivado Design Suite User Guide: Partial Reconfiguration (v2017.1). [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf. Accessed on: Dec. 3, 2017.

[21] S. Liu, R. N. Pittman, and A. Forin, "Energy Reduction with Run-Time Partial Reconfiguration," *in Proc. 18th annual ACM/SIGDA international symposium on Field-Programmable Gate Arrays*, 2010.