Enabling Virtualization on RISC-V Microcontrollers

Stefano Mercogliano, Daniele Ottaviano and Alessandro Cilardo

Department of Electrical Engineering and Information Technologies,
University of Naples Federico II
{stefano.mercogliano, daniele.ottaviano, acilardo}@unina.it

Abstract

This paper briefly provides the main motivations and goals to enable isolation and virtualization on MMU-less devices, introducing a high-level description of an extension meant to support virtualization on RISC-V microcontrollers through a set of new hardware mechanisms, concluding with an outline of our future directions.

Introduction and Context

Nowadays industrial-related research (e.g. Automotive, Aerospace, Nuclear, Military) is going towards the consolidation of multiple real-time applications and OSes on single boards with the main goal of optimizing a set of parameters usually referred to as SWaP-C (Size, Weight, Power, Cost). To this end, embedded virtualization proved to be an effective solution, usually achieved by means of ad-hoc tailored hypervisors. Despite providing powerful virtualization hardware support, application processors (e.g. Arm Cortex-A family) imply non negligible downsides with regard to real-time embedded virtualization. The growing complexity of mechanisms in multicore application processors, such as prefetching, branch prediction, cache coherence as well as memory virtualization, improves the average performance while decreasing predictability and hardening the process of certification. Partitioned hypervisors attempt to solve this problem through static resource allocation, but they eventually run into issues with inefficient resource usage and high power consumption. Microcontrollers are way cheaper than application processors, and buying more of them is the solution mostly adopted nowadays. However, along with the growing demand for numerous integrated functionalities, virtualization would lead to advantages in scalability, ease for software integration, porting and updating along with benefits in terms of reliability and availability.

Microcontroller virtualization is a relatively new concept, yet not unexplored. Pinto et al. [1] used TrustZone-M to enable virtualization on Cortex-M, providing a two guests model. In [2], the authors describe uTango, a secure OS relying on TrustZone-M to isolate trusted applications and OSes in the non secure world. In general, TrustZone-M can be used to support a secure world hypervisor and a set of isolated guests in the non secure world, keeping guest states into the secure memory. RISC-V architecture lacks a comparable stable mechanism, therefore many proposals have been discussed in the community to isolate M mode code [3]. The *Trusted Context Extension* (TCx) and especially the *Trusted Execution State* (TESx) have been designed to mimic and improve TrustZone-M on RISC-V microcontrollers. Differently, *Machine Mode Split Extension* (MMSx) proposes to split the Machine mode in two, similarly to H-extension, to host a separate secure monitor. In the recent Armv8-R specification, Arm was the first to introduce a virtualization hardware support for microcontrollers. The key idea is to add the EL2 for the hypervisor to run and support a second level MPU directly handled by the host to isolate guests. Supporting virtualization on RISC-V microcontrollers with a TrustZone like approach would dictate a set of limitations in terms of performance, predictability, scalability and reliability, mostly due to the dual world model and the focus on security rather than reliability. Therefore, we propose a hardware support with a focus on explicit virtualization rather than an adapted TrustZone-like model.

Virtualization Support Design

Hardware virtualization support has been designed according to the requirements of *Equivalence, Control* and *Performance* described by Popek & Goldberg. However, microcontroller workloads usually demand predictability as well; improving performance mostly consists of new hardware structures shared among guests, causing extra and nondeterministic latency and harnessing predictability (e.g. caches, predictors). In the literature, real-time virtualization has been successfully demonstrated mainly using partitioning-based approaches (e.g. Jailhouse, Bao). Based on this idea, we follow the principle that adding new shared resources improves performance and does not reduce predictability only if such resources can be perfectly, but not necessarily equally, partitioned among all guests owning that resource. For this to happen, the host machine must provide a significantly higher number of resources compared to the guest machines.

Similarly to H-extension and MMSx, we split M mode in a *Root Machine* (RM) mode and an *Emulated Machine* (EM) mode. This implies the duplication of M-CSRs. However, differently from H-extension, the RM mode shall not support type-2 hypervisors nor handle page tables. RM-mode can access guest CSRs and handle a set of new RM-CSRs (e.g. guest and interrupt handling). EM mode must see EM-CSRs like native M-CSRs in order to let guests run unmodified. A consequence of M mode virtualization, compared to S mode virtualization, is the increase of guest context. Therefore, along with the need of avoiding nondeterminism, we must support a set of mechanisms based on the partitioning principle outlined above to minimize context size and switch overhead.

We assume a number of guests per core less than or equal to 8, so that the partitioning principle can be reasonably applied and its related mechanisms programmed by means of RM-CSRs. *General Purpose* (GP) registers can be *totally banked* for each guest with a very small N (e.g. $N = 2$). In case of E-extension enabled guests the hypervisor might partition a single GP regfile in two parts (i.e. GP coloring). However, because the GP regfile accounts for a significant percentage in terms of both area and power consumption in microcontrollers, we inherit the idea of TESx and TrustZone-M to provide *partial banking* (i.e. stack pointer, thread pointer, global pointer). Similarly to Armv8-R, guests isolation can be achieved by adopting a second level of protection. The PMP can be transparently programmed and handled by guests (e.g. to isolate tasks), while the extra protection layer is in charge of preventing a guest to tamper with another guest. Such a choice implies that the PMP is accessed by each guest context. In order to alleviate such a burden, the hypervisor can program a *PMP coloring* mechanism so that each running guest only sees a fraction of the pyhsical PMP. This idea is a specific instance of a CSRs partitioning mechanism introduced in [4], although implementation details are still to be defined. One key observation is that despite being effective in improving context switch cost, coloring techniques are not always applicable, since they rely on specific configuration constraints. Therefore, we are considering the introduction of new instructions to quickly context switch depending on the enabled performance features (e.g. colored resources, TCMs). Moreover, in order to properly delegate interrupts to guest, the RM mode must hold a RM-CSR for the current running guest ID, which must be propagated on the fabric as well in order to be shared with peripherals and interrupt controllers (i.e. CLIC, PLIC).

One major problem underlying unmodified execution is that binaries are written to run on a specific platform physical memory layout. Because no virtual memory is supported, it is necessary to relocate guest addresses to host addresses in a unique and transparent way. Such a mechanism does not require intermediate physical pages to be swapped or revoked, as it would result in unpredictable memory accesses, but would still require some intelligence, yet simple and minimal. In [5] Pan et al., describe a technique to generalize MPUs in a guarded-radix-tree-based structure, providing an optimized MPU configuration at linking time; although loosely coupled with the memory relocation problem, it inspired our design of a *Phyisical Memory Relocation(PMR) unit* to support a lightweight memory relocation mechanism to provide transparent unmodified execution.

Conclusions and Future Works

This document briefly describes the underlying motivations towards embedded virtualization and the limitations for modern RISC-V based microcontrollers. We also provide a high-level description of our ongoing project; we are now working on the mechanism of memory relocation and context switch instructions, while timer and peripheral handling still remains an open problem. In the short run we expect to provide a functional proof-of-concept and some experimental results targeting the cores from the PULP family, in both single- and multi-core configurations, adopting a simulated approach using Verilator and FPGA synthesis. For the future, we are motivated in refining the architecture and building an open software environment, including a hypervisor, capable of fully exploiting the proposed extension.

References

[1] Sanndro Pinto et al. “Virtualization on TrustZone-enabled microcontrollers? Voilà!” In: *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2019, pp. 293–304.

[2] Daniel Oliveira, Tiago Gomes, and Sandro Pinto. “uTango: an open-source TEE for IoT devices”. In: *IEEE Access* 10 (2022), pp. 23913–23930.

[3] Stefano Mercogliano. *RISC-V Machine Mode isolation Proposals Overview*. URL: `https : / / docs . google . com / presentation / d / 1aMv4zJ6qu4teE _ 1lGs0Xr59pgpe04erT2JBRF4R1TGM/edit#slide=id.p`.

[4] Alessandro Cilardo and Stefano Mercogliano. “Flexible privilege management for microcontroller-class RISC-V cores”. In: *Microelectronics Reliability* 137 (2022), p. 114771.

[5] Runyu Pan et al. “Predictable virtualization on memory protection unit-based microcontrollers”. In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2018, pp. 62–74.