

Understanding the Differences Between Armv8-R and Armv8/9-A Virtualization

Contributors:

Dr. Carmelo Loiacono

Technical Product Owner, Green Hills Software

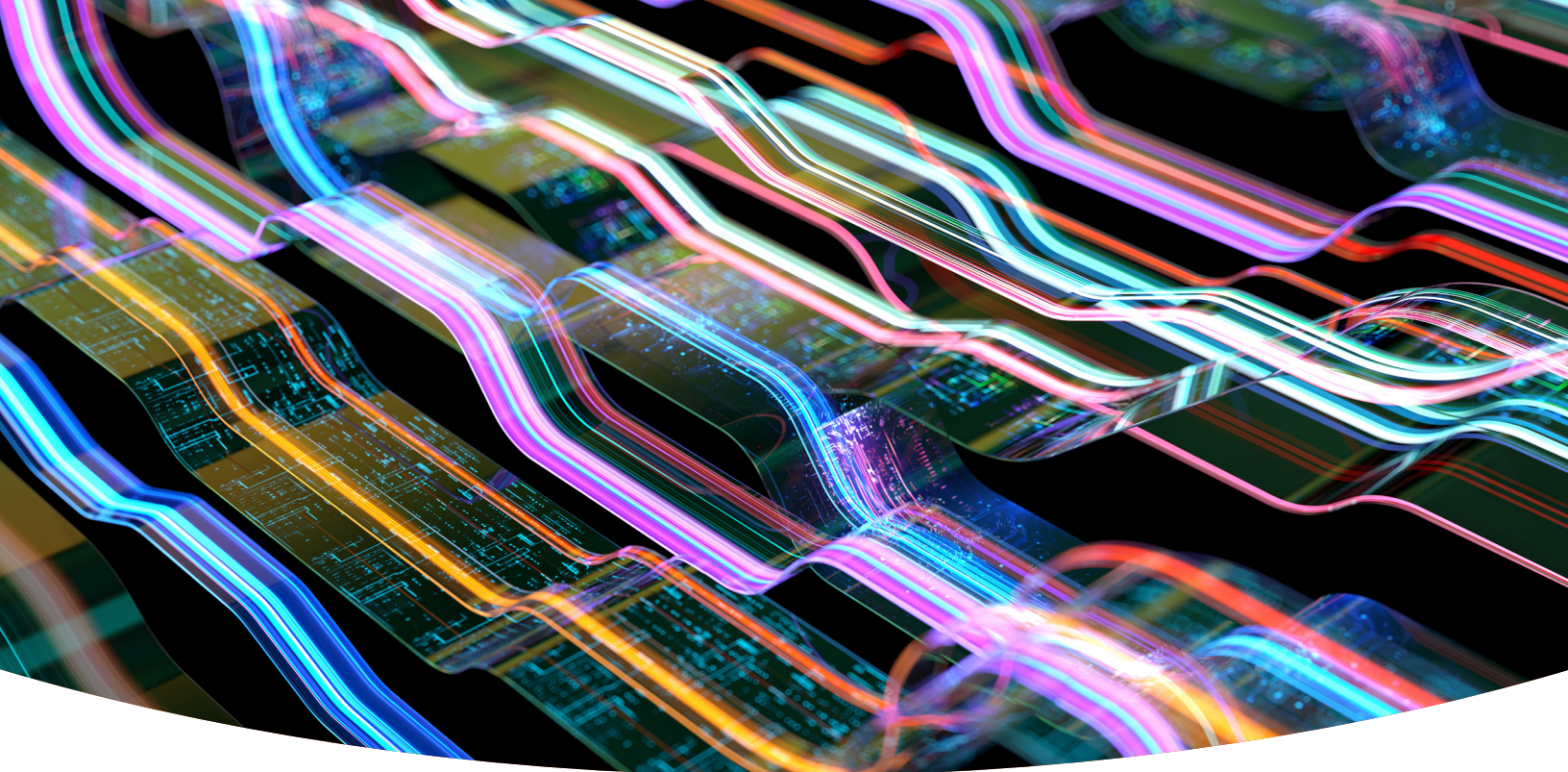
Arnaud Van den Bossche

Director Business Development, Green Hills Software

Bernhard Rill

Director Automotive GTM EMEA, Arm





WHITE PAPER

arm

EXECUTIVE SUMMARY



As the automotive industry moves toward software-defined vehicles (SDVs) and centralized E/E architectures, the need for efficient, scalable, and secure computing platforms has never been greater. Virtualization plays a key role in this transformation by enabling multiple operating systems to run on a single hardware platform, allowing function consolidation, cost reduction, and enhanced security.

While virtualization is well established in application processors, its adoption in real-time microcontrollers (MCUs) presents unique challenges due to the need for deterministic execution, real-time constraints, and hardware limitations. This white paper explores the differences between Armv8-R and Armv8/9-A architectures in the context of virtualization, hypervisor implementation, and system consolidation.

Virtualization enables the consolidation of multiple workloads—from infotainment systems and Advanced Driver Assistance System (ADAS) to motion control and vehicle safety functions—onto a single processing platform. This leads to:

- Reduced Hardware Costs: Fewer physical Electronic Control Unit (ECU) mean lower material and integration costs.
- Enhanced System Flexibility: Software updates and over-the-air (OTA) updates improve longevity and maintainability.
- Enhanced Safety and Security: Virtualized separated environments provide controlled isolation, ensuring that failures in one domain do not impact others.



Introduction

With the growing interest in autonomous vehicles and connected cars, embedded systems are becoming increasingly complex and interconnected. There is a pressing need for more efficient, scalable, and secure solutions as traditional embedded systems, often built on purpose-specific hardware, struggle to keep up with these demands given their inflexibility and high maintenance costs.

Virtualization allows multiple operating systems to run on a single hardware platform and also multiple applications, possibly on multiple instances of the same OS for better separation. This capability is crucial for modern applications comprised of mixed-criticality functions, especially in the automotive and industrial sectors, where upgradability, updatability, and over-the-air (OTA) updates are essential.

By consolidating several functions into a single System on Chip (SoC), Tier 1 suppliers and OEMs can streamline applications like Digital Cockpit, ADAS, electrification, motion control, and body control. It also helps reduce hardware costs, simplifies system management, and lowers energy consumption. Additionally, virtualization allows to reduce the cost of maintaining ASIL certification in system running functions with different safety integrity levels. Finally, it helps enhance security and isolation, which are paramount in applications like automotive safety systems. One key mechanism to enable this consolidation and integration trend are automotive grade hypervisors.

Use Cases for Virtualization on Microcontrollers and Microprocessors

Microcontrollers (MCUs) and microprocessors or System on Chips (SoCs) serve different purposes and have distinct requirements. Understanding these differences is critical to leveraging virtualization effectively.

Microcontrollers are tailored designs for real-time and safety-critical systems, such as automotive, industrial automation, and robotics. Virtualization enables the simultaneous execution of real-time and deterministic functions alongside different guest operating systems and their application functions.

Key microcontroller components include:

- MPU (Memory Protection Unit): Uses pure physical addresses with a small, fixed number of ranges.
- Single core or Multicore: In case of Multicore often AMP (Asymmetric Multiprocessing) with one OS per core.
- Safety: Capable of achieving ASIL-D safety levels with hardware support like lockstep cores, separate power domains and additional hardware monitoring features (memory error correction coding ECC)
- Privilege mode differences: Normally only reflect two modes in 32-bit CPUs (e.g. Armv7-M, Armv8-M, Armv7-R) – Supervisor and User mode (EL0 and EL1, EL means Exception Level), and in addition EL2 in the case of Armv8-R

Microprocessors (application processors) are designed for general-purpose computing, automotive in-vehicle infotainment (IVI) and ADAS, mobile devices, laptops servers, and high-performance applications. Those SoC designs normally include a primary compute including one or several clusters of Cortex-A processors (thus Armv8/9-A CPUs), and a safety island built with real-time deterministic processor (pre-dominantly Armv8-R based). Its primary compute virtualization offers more capability and complexity and is aimed at supporting multiple operating systems and applications with complex memory management and security requirements, and peripheral sharing with the key components being:

- MMU (Memory Management Unit): Uses virtual addressing with a large number of pages. Within the MMU the page sizes are predefined (e.g. 4K, 2M, 1G)
- Multicore: Often SMP (Symmetric Multiprocessing) with one OS for all cores and AMP for the safety island and infrastructure components (security, low-power management and system control).
- Privilege mode differences: up to four modes labelled EL0 to EL3 (on the application cores)
- Safety: Typically up to ASIL-B, with software support for features like software test libraries, time supervision, deadline monitoring and critical application monitoring.

Consolidating multiple functions into a single SoC has several benefits, including fewer SoCs, common resources, and easier upgradability and OTA updates. However, it also introduces complexity, which can be a risk for safety and cybersecurity and in turn - to introduce variety need to be addressed with a certified separation / microkernel real-time operating system and an intelligent system architecture.

VIRTUALIZATION FOR EMBEDDED SYSTEMS

Within Automotive, virtualization technology is used on application Cortex-A processors for two generations of Automotive platform development starting with the Cortex-A15. Virtualization technology has already been deployed for two decades in embedded systems like avionics and the military.

While virtualization is not new for embedded systems, it is relatively new for high-end real-time microcontrollers in the automotive space. However, real-time microcontrollers have unique architectural features, such as constrained hardware resources, real-time response requirements, and special boot-up demands which then need to be addressed in the overall system architecture.

Some examples of high-end real-time microcontrollers which offer support for embedded hypervisor software include the [NXP S32Z/E](#) family of processors and the [ST Microelectronics Stellar P/G](#) integration MCUs.

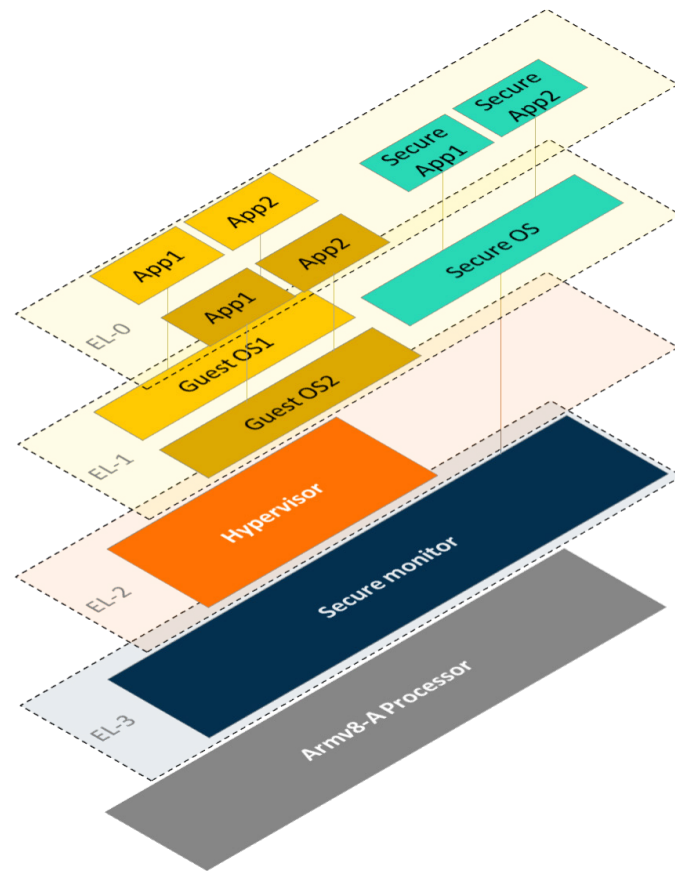
“While virtualization is not new for embedded systems, it is relatively new for high-end real-time microcontrollers in the automotive space.”

Differences Between Armv8/9-A and Armv8-R Virtualization

Armv8/9-A Cores (e.g. Cortex-A76AE, Cortex-A78AE, Cortex-A720AE and V3AE):

The architecture of Armv8/9-A cores is as follows:

FIG. 1
Armv8/9-A architecture
exception levels



— MMU: Provides support for virtual addressing

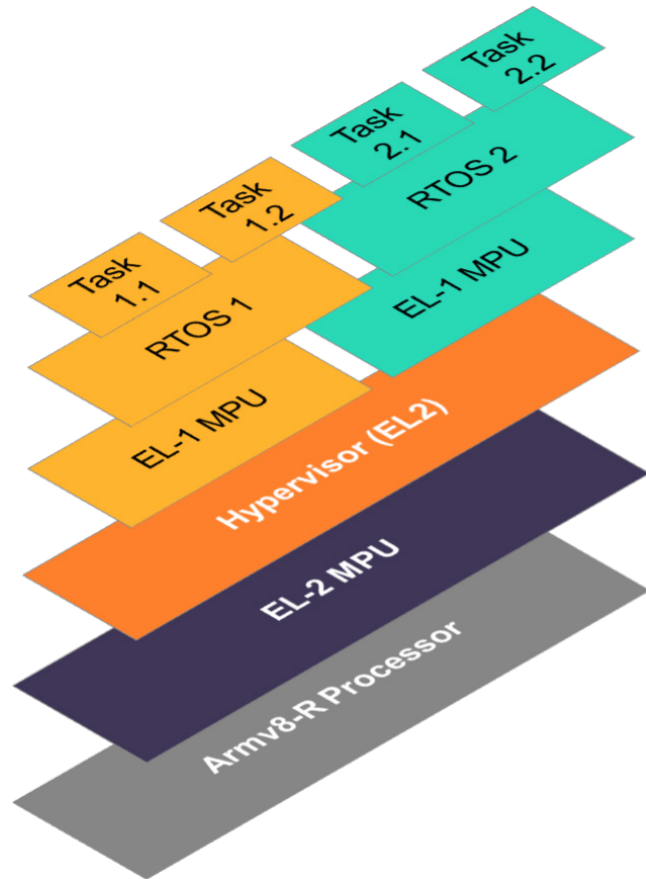
- The hypervisor will map the IP of guests accessing the memory / peripheral device to the real physical address of the SoC
- The guest OS believes it is talking directly to hardware, but actually the hypervisor is ensuring that the memory/peripheral accesses are mapped to the correct physical addresses
- Advantage of this approach is that the guests' operating systems can run (normally) unmodified, but the additional hardware synchronization mechanisms to ensure coherency imply that the systems are not fully deterministic and no longer operates in real-time

-
- Virtual Interrupts and Timers:
 - Key components for hypervisors to orchestrate the integration of different guests' operating systems in several virtual machines
 - On application cores the device abstraction and configuration / access strategy is defined through the OASIS VirtIO standard.

Armv8-R Cores (e.g. Cortex-R52+ and Cortex-R82AE):

The architecture of Armv8-R cores is as follows:

FIG. 2
Armv8-R architecture
exception levels



-
- MPU: Provides only physical addressing
 - This architecture implies that there is NO address translation, so the guests need to be aware at build time which exact addresses of the memory and the peripheral devices they are eligible to access (and in what context, thus exception level)
 - For this instruction set architecture the guest operating system needs to be adapted so that it only accesses memory and peripheral devices based on the system software architecture definition.
 - Advantage of the approach is that a deterministic software execution still can be guaranteed.
 - Virtual Interrupts and Timers:
 - Similar to Armv8/9-A but tailored for real-time applications (e.g. around determinism).
 - Device virtualization principles for sharing memory and peripherals for the Armv8-R profile is described in the Armv8-R device virtualization principles [white paper](#).

KEY FUNCTIONALITIES OF A HYPERVISOR:

Hypervisors fulfill dedicated tasks within the overall software system architecture:

- Isolation or separation:
 - Capability to support partitioning of resources
 - Prevents interference between applications (and thus allows the fulfilment of each VM and System wide safety and security requirements)
 - Protects resources associated to each application
 - Allows each virtual machine to run its own operating system
 - Can act as an operating system for operating systems
- Virtualization
 - Gives VMs direct access to hardware
 - If needed, optimizes the usage of resources by sharing them between applications (this includes sharing mechanisms in hardware and in software)
- Boot-up and run-time orchestration
 - Overall system configuration
 - Power-up behaviour
- Low-power mode activation

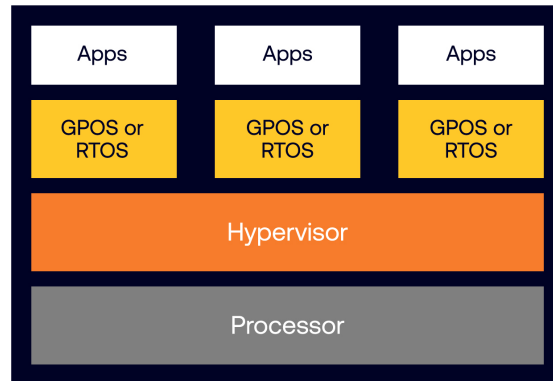
Types of Hypervisors

There are different types of Hypervisors which can be categorized in separate architectures:

— Type 1:

- These hypervisors are also suitable for MCUs with limited resources
- Requires two context switches for critical tasks upon receiving an interrupt:
 - #1 interrupt is passed into the hypervisor
 - #2 interrupt is into the guest operating system for task scheduling
- General Purpose Operating System (GPOS) runs isolated next to RTOS

FIG. 3
Type 1 hypervisor
architecture



— Type 2:

- A Type-2 Hypervisor is only as efficient, reliable and safety-certifiable as its Host OS (usually Linux). Based on this a Type 2 hypervisor built on a non-safety OS is normally not suitable for embedded devices
- Requires one context switch for critical tasks upon receiving an interrupt
- Run High-Level Operating System (HLOS) isolated to protect critical execution

FIG. 4
Type 2 hypervisor
architecture

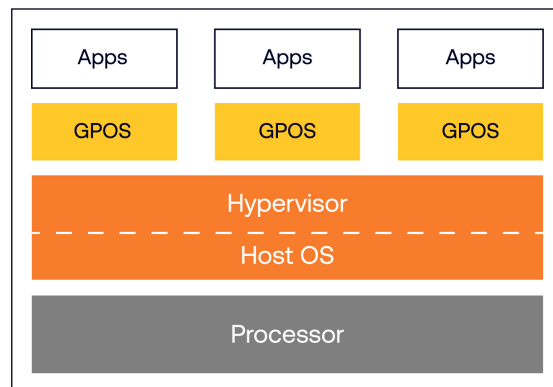
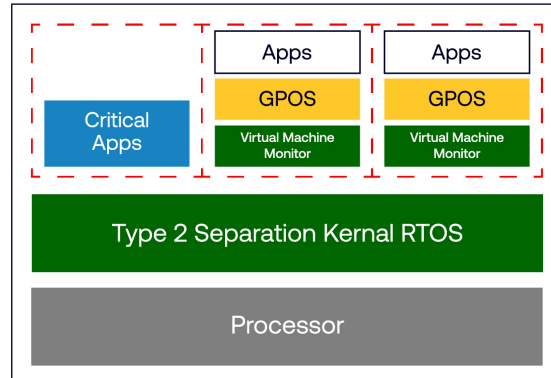


FIG. 5
Type 2 microkernel
architecture

- Type 2 Microkernel:
 - Combines the benefits of Type 2 with microkernel architecture features, providing additional isolation and scheduling features next to the hardware enablement)



TASKS OF AN EMBEDDED HYPERVISOR RUNNING ON AN MCU:

In the case of Armv8-R when a system software architecture includes an exception level 2 hypervisor there are several basic operations which need to be addressed by the separation kernel / embedded hypervisor:

- Boot up orchestration
- System configuration to enable virtualization
- Control of power-up and resetting of cores
- Clock configuration
- EL2 MPU Configuration
- Assigns memory and peripheral access to guests
- Grants hardware access rights
- Assigns CPU time to guests according to static configuration
- Limit access to important hardware components (e.g. via the EL2 MPU)
- Interrupt management
- Booting of each guest
- System supervision and error management

SoC/MCU specifics

Next to the Arm architecture / CPU specific dependencies there are silicon vendor integration specific aspects which will need to be reflected and supported by the embedded hypervisor. For example, the high-end integration microcontroller Stellar SR6 from ST Microelectronics add hardware features that are useful for embedded hypervisors to safely separate and share system resources.

- 1st of all, it uses the foundation from the Arm Cortex-R52:
 - EL2 MPU to provide spatial separation between virtual machines
 - Thus the MPU controls the read/write access at the local resources level
 - GICv3 interrupt virtualization
 - Forwarding of hardware interrupts
 - Injection of virtual interrupts
 - Virtual Machine ID (VMID) support
 - Arm generic timer access
- The silicon designer ST Microelectronics have then added to the Arm CPU the following components to embed the Cortex-R52
 - Peripheral bridge
 - The peripheral bridge controls the read/write access at the peripheral resource level
 - Network-on-Chip (NoC) firewall
 - The NOC firewall controls the read/write operations at the system resource level

EXAMPLE CONFIGURATIONS OF GREEN HILLS SOFTWARE PRODUCTS E.G. FROM SPECIFIC SOC/MCU

Green Hills offers two hypervisors:

- for the Cortex-A and Cortex-R82AE it is the [INTEGRITY Multivisor](#)
- for Cortex-R52 cores it is the [u-visor hypervisor](#)

ARCHITECTURE AND DEVELOPMENT PHASES (INCLUDING TOOLING REQUIREMENTS):

In the Architecture and Development phases to port software to a target silicon platform there are several key configuration topics which need to be considered if guest operating systems are to be used. This is where the hypervisor plays an essential role:

As a starting point it is vital to configure the system and orchestrate the boot-up. The safety use-case is driving this configuration aspect, and due to the differences in the application usage the safety use-case needs to be tailored for each customer and function. What is key is which VMs needs access to which memory regions / peripherals:

- What peripherals are shared and which ones are directly assigned to VMs
- What memory region is shared and which one is not (and if shared which access rights are given: read-only or write)
- And are the above topics steered and/or controlled by the hypervisor

Once configured it is necessary to have efficient means of debugging virtualized workloads, kernels, device drivers and application code:

- Debugger operating system and hypervisor awareness
- System clock dependencies and the opportunity to map all individual traces to one system clock base
- Unified view of these software layers

To further optimize the system performance it is vital to factor in the efficiency and overhead of a hypervisor and the guest operating systems:

- A balanced view of microscopic benchmarks versus system-level KPIs
- Key Performance Indicator (KPI) examples:
 - CPU load impact
 - VM switching time
 - Interrupt latency overhead
 - VM boot-up, exit and restart, sleep and wake-up time
 - Peripheral performance impact (virtualized vs native), applies e.g. for SPI, CAN, Eth, Storage etc.
 - Hypervisor scheduling & memory footprint impact

Once the KPIs are known, the results should lead to performance optimization initiatives, especially as the consolidation and the hypervisor brings new challenges as the more advanced SoC's with more complex software, add a certain risk regarding safety and cybersecurity. To address these risks there are several design and implementation requirements for hypervisors:

- Simple and small (minimization of complexity)
- Provide separation of applications (componentization)
- Distinguish between safe and non-safe executions
- Non-safe executions can only access pre-defined OS objects (least privilege)
- Leverage SoC features for protection, allowing for mixed criticality solutions
- Developed according to safe & secure processes, e.g., ISO26262 ASIL-D, ISO/SAE 21434 (secure development process) including the tooling to configure the hypervisor (and the interfaces linked to it)
- Validated by industry safety and security experts (independent expert validation)

Based on the additional software complexity of the Software Defined Vehicle (which includes the real-time consolidation requirements) there is an absolute need to have state-of-the-art development tools to quickly FIND the root-cause of the bug, optimize the system and solve performance problems and bottlenecks

The tool must provide the following:

- Easily measure interrupt latency, context switch time, and boot time with great accuracy and precision
- See system behavior including visualization
- Log custom data values from the application
- Operating system agnostic, processor agnostic, trace log agnostic
- Highly optimized for minimal system intrusion

Conclusion

Virtualization for automotive use-cases is not new, but its deployment to real-time microcontrollers is. The use cases and workloads differ significantly between application processors (Armv8/9-A) and real-time processors (Armv8-R), requiring the right strategy for using virtualization and hypervisors. Arm core architectures are optimized for virtualization, and Green Hills hypervisors are tailored to leverage these features, providing a robust solution for developing, optimizing, deploying, and debugging systems using hypervisors on microcontroller and application processors.

References

- [1] Arm A-profile A64 Instruction Set Architecture
<https://developer.arm.com/documentation/ddi0602/2024-12/?lang=en>
- [2] Arm R-profile Instruction Set Architecture
https://developer.arm.com/documentation/DEN0130/0100OASIS_VirtIO
<https://groups.oasis-open.org/communities/tc-community-home2?CommunityKey=b3f5efa5-0e12-4320-873b-018dc7d3f25c>
- [3] Armv8-R device virtualization principles white paper
<https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/device-virtualization-whitepaper.pdf>
- [4] Armv8-R software integration evolution community blog:
<https://community.arm.com/arm-community-blogs/b/automotive-blog/posts/software-integration-on-armv8-r-cortex-r52-plus>
- [5] Green Hills INTEGRITY Multivisor
https://www.ghs.com/products/rtos/integrity_virtualization.html
- [6] Green Hills μ -visor
<https://www.ghs.com/products/u-visor.html>