

Timing Analysis for Hypervisor-based I/O Virtualization in Safety-Related Automotive Systems

Andre Kohn and Karsten Schmidt
AUDI AG

Jochen Decker
Easycore GmbH

Maurice Sebastian
Symtavision GmbH

Alexander Züpke
RheinMain University of Applied Sciences

Andreas Herkersdorf
Technical University of Munich

ABSTRACT

The increasing complexity of automotive functions which are necessary for improved driving assistance systems and automated driving require a change of common vehicle architectures. This includes new concepts for E/E architectures such as a domain-oriented vehicle network based on powerful Domain Control Units (DCUs). These highly integrated controllers consolidate several applications on different safety levels on the same ECU. Hence, the functions depend on a strictly separated and isolated implementation to guarantee a correct behavior. This requires middleware layers which guarantee task isolation and Quality of Service (QoS) communication have to provide several new features, depending on the domain the corresponding control unit is used for. In a first step we identify requirements for a middleware in automotive DCUs. Our goal is to reuse legacy AUTOSAR based code in a multicore domain controller. In an example use case scenario, we analyze the end-to-end latency for transmitting and receiving CAN messages in a system using a hypervisor-based virtualization approach. A model-based timing analysis and an implementation on a state-of-the-art automotive microcontroller shall help to assess the usability for safety-related domains. Additionally, the timing behavior is compared to a common AUTOSAR implementation. We could show that a well-configured hypervisor and scheduling can provide similar results as an AUTOSAR implementation under certain circumstances. Furthermore, a model-based timing analysis allows evaluating different hypervisor configurations without physical hardware.

CITATION: Kohn, A., Schmidt, K., Decker, J., Sebastian, M. et al., "Timing Analysis for Hypervisor-based I/O Virtualization in Safety-Related Automotive Systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 10(2):2017, doi:10.4271/2017-01-1621.

INTRODUCTION

The increasing communication effort and data exchange of today's and future electronic vehicle components result in a considerably higher software complexity. This leads to a huge change in the automotive E/E architecture which is directly associated with numerous new challenges. The requirements for ECU architectures concerning real-time capability, CPU performance and memory management significantly differ, depending on the considered vehicle domain. However, isolation is one key aspect which is relevant for each domain and has to be considered in nearly all current and future automotive domain controller designs.

Generally, multicore CPUs provide more performance and are able to consolidate software of different Automotive Safety Integrity Levels (ASILs) on a single system. Another goal when applying highly integrated ECUs is to reuse legacy code of existing singlecore ECUs while minimizing the modification effort. This means, that different operating systems might be run on the same controller. All these use cases imply the need for an efficient isolation concept to avoid mutual interferences.

The controllability of highly-integrated multicore domain controllers can be improved by a well-conceived middleware that manages the access to resources and peripherals. An established solution, especially in the IT domain is a hypervisor-based system-virtualization. In

contrast to embedded microcontrollers, the IT domain uses microprocessors which provide hardware-support for virtualization. Due to the fact that most hardware architectures for safety-related automotive electronic systems, do not provide such a hardware support, only software-based virtualization approaches can be considered. Previous implementations are mostly based on para-virtualization because the hardware architectures do not fulfill the requirements for a full-virtualization of the Popek & Goldberg theorem [1]. Even though virtualization provides several advantages such as a strong isolation of virtual machines (VMs), it is still very uncommon in automotive ECUs. This is often due to the disadvantageous overhead in timing and memory resources

In our contribution, we analyze the timing behavior respectively the timing overhead of the ecHypervisor by Easycore GmbH. The evaluation target hardware is a NXP/Freescale MPC5748G CPU using the PowerPC architecture. Our goal is to consolidate tasks from AUTOSAR-based ECUs from the chassis domain which communicate with each other but also with an ECU from the motor domain. Here, more than one AUTOSAR system runs in parallel on the target hardware

In the second section we present similar approaches and evaluations of related publications. Section 3 gives an overview about requirements for system design and the architecture of target platform. The implementation and applied hardware and software architecture is described in Section 4 before we discuss our evaluation results and conclude in Sections 5 and 6.

RELATED WORK

System virtualization is well established in different software domains to isolate and separate several applications. Desktop and server-based systems often use hardware support of the processor architecture such as Intel VT-x and AMD-V which allow implementing full-virtualization and hence do not require a modification of the guest systems. A similar approach for embedded systems was presented by ARM with the introduction of the ARMv7 architecture. Controllers which use this architecture can often be found in smartphones and tablet PCs. Another field of application for these CPUs are automotive infotainment systems which benefit from the increased performance and the ability to isolate different operating systems. However, virtualization is a new field of application for automotive electronics so that implementations are rare.

Generally, most available hypervisor solutions for embedded systems use a microkernel. Popular approaches are the Sysgo PikeOS, GreenHills Integrity, WindRiver Hypervisor and the L4-based OKL4.

The features of PikeOS were evaluated by Continental AG who built an In-Vehicle Infotainment (IVI) demonstrator which combines an AUTOSAR system with a POSIX-based security stack. Additionally, the Continental-specific Open Infotainment Platform (OIP) which uses GENIVI and a partition running Android with custom-oriented apps were implemented. The operating systems either use Asynchronous Multi-Processing (AMP) with several partitions on one core or Synchronous Multi-Processing (SMP) with only one

partition on the same core. The implementation runs on an ARM Cortex-A9 processor while a port for the Cortex-A15 providing hardware support for virtualization is still under development. This demonstrator shows a possible software architecture with system virtualization for the infotainment domain [2].

In the context of the ARAMiS project, an automotive demonstrator was developed to evaluate the RTA-HV, a hypervisor by Etas Ltd., for the applicability in highly-integrated automotive ECUs. The demonstrator uses the Infineon TriCore architecture, more precisely the AURIX TC277, running two individual automotive software systems with AUTOSAR Basic Software and AUTOSAR-OS each. The first system consisted of the BMW AUTOSAR Core (BAC) with different applications, while the second system runs a powertrain application [5].

A continuative work was the development of a cost model to determine the workload, resource utilization and timing behavior of two implementations. The first one is based on an intrusive approach with an AUTOSAR-only system, while the second, non-intrusive implementation applies the RTA-HV. A disadvantage of the hypervisor was that only one partition can be embedded on the same core [4].

Virtualization of a CAN controller and a mechanism called pre-virtualization were realized on the Infineon AURIX TC27x in another contribution. Pre-virtualization combines the advantages of a full-virtualization approach with the efficiency of a para-virtualization approach. The efficiency improvement is achieved by applying binary translation before the final implementation. The comparison of the clock cycles and execution time of the virtualized instructions show the efficiency of this approach. Furthermore, the work lists general requirements for a hypervisor from the automotive point of view [3].

An AUTOSAR-compatible microkernel was presented by Elektrotbit Automotive GmbH. The authors came to the conclusion that the kernel performance is comparable to that of a standard operating system [6].

REQUIREMENTS OF AUTOMOTIVE SOFTWARE ARCHITECTURES

Functional safety is the most relevant aspect for domains with classic automotive applications, e.g. chassis and powertrain. Hence, highly-integrated ECUs have to comply with the requirements of the ISO 26262 standard, namely freedom-from-interference and hard real-time constraints. Furthermore, aspects such as static memory allocation and scheduling have to be considered. This means, distribution of memory must be applied in the design phase, while during execution, memory accesses are protected by a Memory Protection Unit (MPU). In the last years, AUTOSAR has become the standard for software architectures in such safety-related domains due to its static implementation approach. Hence, only hardware architectures with an MPU are supported which means dynamic

memory allocation is not possible. Although, gateway ECUs mostly do not necessarily execute applications with hard real-time requirements, AUTOSAR is used in these control units as well.

Automotive Middleware

An ECU software architecture consists of an application layer, a middleware and the basic software. The middleware is a more generic designation of a runtime environment and coordinates the complete communication and data transfer between applications and also the access to peripherals. This includes sensors, actuators and memory registers. Depending on the domain, the middleware can be configured and adapted to the user's needs while in the automotive domain AUTOSAR RTE is a well-established standard.

Furthermore, the middleware usually offers services and capabilities to virtualize shared hardware and software resources. This allows to encapsulate functions to avoid interferences and to port legacy code from a singlecore to a multicore system. Especially applications running on other operating systems such as Linux or Android often have to be isolated from the rest of the system.

From the application software perspective, the middleware must be able to trigger the application software respectively the user functions. The segregation in space and time but also the management of memory and communication must be handled accordingly. This implies communication between applications on the same ECU and with peripheral to realize a data exchange with other ECUs. Furthermore, an inter-core communication mechanism has to be implemented in multicore CPUs. In case of using more than one operating system on the same CPU, virtualization is a promising approach. Further non-functional requirements for a middleware on nearly all automotive multicore ECUs are the following:

- Guarantee of real-time and performance requirements
- Realization of a safe and secure system
- Handling of multiple operating systems / multi-core capable OS
- Handling applications of different ASILs
- Management of shared resources
- Support of fail-operational
- Support for legacy code applications

In general, hypervisor solutions are less popular for AUTOSAR-based real-time domains and there are only few approaches already available. Furthermore, the approaches are limited and optimized to a specific architecture while a port to another architecture is a cost-intensive task. The virtualized system is often limited by the fact that the hypervisor does not allow implementing more than one partition on the same core. Multiple partitions on one core require an additional effort for temporal separation because each guest system applies its own scheduling policy which has to be synchronized to the other guests. So, scheduling and memory management are key aspects in the system design. Thus, our goal is to use more than one partition on one core while applications functionality must not be influenced.

Temporal and Spatial Separation in Multicore Systems

The implementation of several partitions on a singlecore CPU or on a dedicated core of a multicore system requires an exact and sophisticated system design concerning timing and memory allocation.

Generally, for system virtualization in a multicore system, a partitioned scheduling is an appropriate method to statically allocate tasks during system design. Each guest system respectively virtual machine uses its own operating system and scheduling algorithm. Thus, a virtual machine consists of a taskset and the corresponding scheduling policy which is defined by the guest operating system. If the guests have different requirements for functional safety, meaning different ASILs, the corresponding access rights and priorities have to be allocated to the virtual machines. Hence, virtual machines have different quality of services and can be activated, deactivated and rebooted by the hypervisor. For virtual machine scheduling implemented by the hypervisor, each VM needs a minimum of CPU resources to run.

Due to the fact that virtual machines are handled as operating system tasks, a second scheduling layer is necessary. Hence, a two-level scheduling scheme is applied. The first level is part of the hypervisor and schedules virtual machines while the second level is used within the virtual machines. For the former, a static time partitioning can be used so that each virtual machine has its own time slot for operating. Depending on the priority, the time slot length can be individually defined. An example for an implementation with two VMs, each running two tasks, is illustrated in [Figure 1](#).

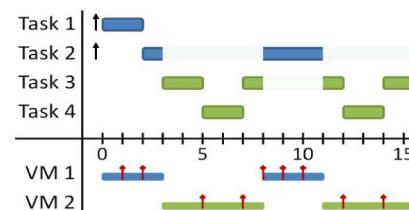


Figure 1. Example for hierarchical scheduling with time partitioning

Generally, schedule design is a complex and time-consuming challenge. An optimal VM scheduler is exactly synchronized with the run-time of the guest system tasks and the task cycle.

As an alternative approach to time partitioning, the required bandwidths of a task and a virtual machine can be calculated. This approach can be adapted to the bandwidth requirements of a virtual machine with its cyclic taskset and scheduling algorithm. With the help of the appropriate function, the required resources of the VM can be derived [8]. Furthermore, a periodic resource model helps to allocate specific timing budgets of a resource so that the maximum resource consumption can be determined. A prerequisite of such a model is that guest system tasks still can meet the defined real-time requirements. This means, the requested CPU time of a virtual machine must always be same or equal as the cumulative CPU time which can be used of the VM [9]. Hence, a mapping of several virtual machines to cores is only possible in a powerful system which provides a suitable VM scheduling

Generally, the main challenges of virtualization in multicore systems are the partitioning and the allocation of virtual machines to cores and the hierarchical scheduling on separate cores.

In addition to temporal separation, an important aspect for a static system is the independency of virtual machines. Shared resources should be limited to CPU and communication bus accesses. Thus, only static and exclusive allocation of memory should be taken into account which is hard to be realized by a memory management unit (MMU) using a dynamic page table. However, an MMU can be configured to use static page tables as well but unfortunately, there is no MMU-based controller available which is certified to execute functions which require an ASIL-D.

Hence, spatial separation respectively memory management in safety-related vehicle ECUs with ASIL-D functions can only be realized by an MPU. The memory is statically partitioned and allocated by the linker at compile time so that during run-time no changes have to be made. Moreover, an MPU is much cheaper to implement which results in reduced costs for a complete ECU.

APPROACHES FOR VIRTUALIZATION IN EMBEDDED AUTOMOTIVE SYSTEMS

The microcontrollers used in automotive ECUs mostly do not allow a full-virtualization approach due to sensitive instructions which can be called from the user mode instead of the full-blown supervisor mode. The unmodified operating systems within the virtual machines require the supervisor mode for hardware access and hence a superior hypervisor mode is necessary to run several independent operating systems on the same hardware in parallel. This mode is implemented by hardware-support for virtualization such as Intel-VT-x and AMD-V or in the ARM Cortex-A family. In contrast, automotive microcontrollers do not provide such a virtualization support so that the guest systems have to be modified accordingly.

Para-virtualization

Today, para-virtualization is often the only approach to benefit of virtualization in embedded microcontrollers while commercial products are rare. Furthermore, operating systems which require an MMU cannot be executed in a virtual machine implemented on safety-related controllers with an MPU. The execution of an AUTOSAR partition in parallel with an Android system is hence not possible. Instead, several AUTOSAR guest systems, each with a different revision, are considerable. In a para-virtualized system, critical instructions are replaced by hypercalls which directly call the hypervisor. In contrast to trap-and-emulate virtualization, para-virtualization reduces run-time overhead by omitting the costly trapping and code checking at run-time.

In a virtualized multicore system, either a single virtual machine or more than one VM could be executed on the same core. The latter case requires a second scheduling level in the hypervisor and as an addition a core scheduler which manages access to the virtualized resources. The effort to synchronize both schedulers is a huge challenge while the complete system becomes much more complex.

Microkernel

Generally, a hardware architecture provides a privileged and an unprivileged mode while the kernel, as a central component of most operating systems, runs in the privileged mode. Hence, it has full access to the hardware, while applications execute unprivileged and have limited access rights. Common desktop OSes typically use a monolithic kernel where all OS services such as memory management, process management, drivers and the file system execute in the privileged supervisor mode. This results in a big Trusted Code Base (TCB) and a fault in a driver could crash the whole system.

In contrast, microkernels use only a minimal privileged software layer which provides general methods concerning OS services and user applications. They provide address spaces as containers similar to page tables to control memory accesses. Furthermore, microkernels use scheduler activations or threads to abstract execution time. And lastly, microkernels provide means for communication between executing entities

In a microkernel implementation, the software resources are separated into kernel mode or user mode. Generally, a microkernel layer can be used as a hypervisor to prevent the direct access to the hardware resources. The software processes are encapsulated in their own hardware address while the communication with other parts of the system is only possible by invoking kernel mechanisms.

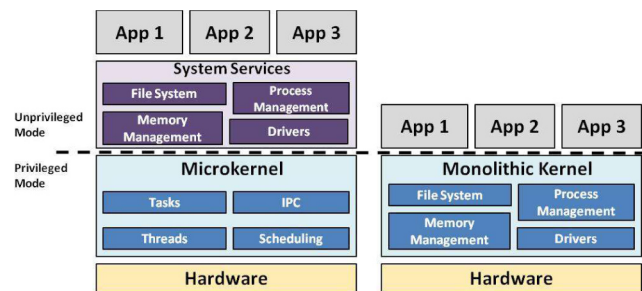


Figure 2. General structure of a microkernel and monolithic kernel

A hypervisor can either be based on a microkernel or on a monolithic kernel. The latter implements memory management mechanisms, scheduling as well as the device drivers and hence has a big TCB which is ineffective in real-time systems. Instead, a microkernel-based hypervisor is much smaller and implements only the basic functionality while device drivers are moved to the partitions. An advantage is that the drivers are only used in those partitions which use the corresponding resource. Furthermore, the hypervisor layer is minimized which leads to a small TCB. The main architecture differences of a monolithic and microkernel can be found in [Figure 2](#).

AUTOSAR Communication Stack

Generally, AUTOSAR provides the COM basic software module to realize bus communication. Each applied bus technology uses the same protocol stack structure while the AUTOSAR signals are consolidated to signal groups. One or more signal groups form an Interaction Layer Protocol Data Unit (I-PDU) which can be handled by the COM module. The I-PDU length depends on the maximum