Siemens Digital Industries Software

Using hypervisor for infotainment and AUTOSAR consolidation on a single ECU

Executive summary

Current approaches used to tackle the complexities in cockpit domain units are both cost prohibitive and lacking in performance. Utilizing virtualization in automotive software architecture provides a better approach when taking on these complexities. This can be achieved by encapsulating different heterogeneous automotive platforms inside virtual machines running on the same hardware. This approach not only provides a more efficient way of communication and reduces the cost of adding a dedicated micro-controller to each platform. It also reduces the development cost by reusing legacy platforms as encapsulated virtual machines without the need for new adaptation efforts.

Mohamed Mounir
Software Engineer
Siemens Digital Industries Software

Contents

Introduction 3
Current approaches 4
System architecture and methodology 5
Siemens Embedded Hypervisor 7
Virtualization for automotive software 8
System implementation 9
AUTOSAR demonstration 10
AUTOSAR application runtime results 11
Conclusion 11

Introduction

Today's automotive manufacturers are racing to deploy new and innovative functionalities in the modern vehicle with technologies that include the human-machine interface (HMI), cloud-based services, vehicle ad-hoc networks (VANET) and autonomous driving. These new technologies increase the complexity of vehicle's electrical and electronics (E/E) architecture and add new requirements for connected software systems. The number of Electronic Control Units (ECUs) is constantly increasing with over 100 ECUs in a modern vehicle. This forces automotive OEMs to consolidate multiple units into a single, high-computing platform.

While this approach simplifies the networking model of the vehicle, it adds more challenges to the automotive software systems architecture. These same challenges exist in the cockpit domain, appearing in Advanced Driver Assistance Systems (ADAS), infotainment head units, and Telematics (figure 1). To further complicate matters, these software applications now require greater variations in their system requirements in terms of safety, security and connectivity. In-vehicle communications and safety critical requirements often require real-time operating systems (RTOSes), while infotainment applications, which are non-safety relevant, run on Linux general purpose operating systems. The combination of the two OSes enables the heterogeneous nature of these applications.

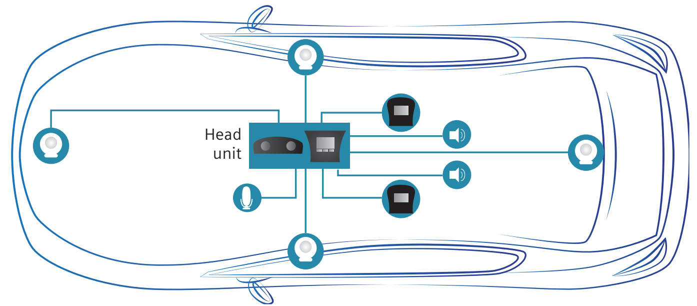

Figure 1. Cockpit domain controller network.

Current approaches

A current approach that might be used to tackle this problem is to add two microcontrollers (MCUs) on the same electronic control unit. One MCU runs the Linux®-based application responsible for high-computational tasks such as AI algorithms and infotainment functions, while the other MCU runs simple, real-time based applications typically used for in-vehicle and diagnostics communication. The two MCUs are connected with a serial peripheral interface allowing communication between these two applications (figure 2).

Although this approach allows for the reuse of standard software architectures, it is inefficient and quite costly to add dedicated hardware for each system. Moreover, it is simply asking too much to have serial communication interfaces to provide reliable communication between these systems. Another approach is to port real-time applications over Linux.

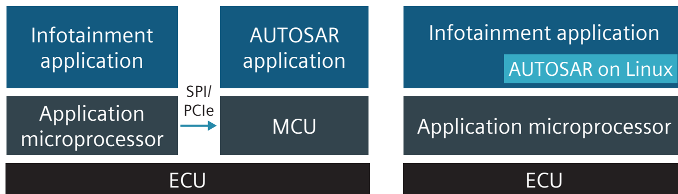

Figure 2. Current approaches for infotainment/AUTOSAR consolidation in the cockpit domain.

It is clear that current approaches to consolidate heterogeneous automotive applications on the same hardware are inefficient. Siemens Digital Industries Software provides multiple products that can be used to address such problems. For instance, Siemens Embedded Multicore Framework can be used to deploy multiple operating systems across homogenous and heterogeneous multicore processors. This solution is based on the OpenAMP standard co-written by Mentor, A Siemens Business and Xilinx. Another approach is to deploy the Siemens Embedded Hypervisor, which uses virtualization techniques to consolidate multiple guest operating systems on the same processor.

This paper presents the concept of virtualization, an efficient alternative automotive OEMs can take when developing heterogeneous automotive applications. The paper demonstrates how Siemens Embedded Hypervisor can be used to consolidate a real-time AUTOSAR application with a Linux-based application. The solution depicted uses the TI Jacinto 6 infotainment evaluation module (figure 3).

Figure 3. TI Jacincto 6 infotainment evaluation module. Copyright® 2019 Texas Instruments, Inc.

System architecture and methodology

This section defines the characteristics of virtualized environments and the methodologies used to satisfy heterogeneous automotive applications requirements.

Virtualized environments

The core of achieving a consistent virtualized environment lies in handling sensitive instructions that affect or depend on the state of the processor hardware. The techniques to achieve this can be summarized as follows:

- **Full virtualization:** This technique depends on a binary translation to enable hypervisor emulation for sensitive instructions. It allows for unmodified guests to run over a hypervisor, but causes performance issues due to the overhead of emulating all sensitive instructions.
- **Para-virtualization:** With this approach, the guest is aware that it runs over a hypervisor and uses specific hypervisor calls for operations involving hardware manipulation. This method eliminates overhead penalties of sensitive instructions emulation, but requires modifying a guest to be able to run over a hypervisor.
- **Hardware-assisted virtualization:** This technique gets the most from full- and para-virtualizations by using hardware extensions to handle sensitive operations, thus removing the overhead of hypervisor emulation in most cases without the need to modify guests to run over the hypervisor. Of course, the drawback to this method is that it only works with modern processors with virtualization support.

CPU virtualization

Arm® TrustZone® is a built-in hardware security solution that defines a security domain consisting of two worlds: secure and non-secure. A processor mode is introduced to monitor world switching, and a privileged instruction (Secure Monitor Call) is introduced to bridge software stacks of the two worlds (through monitor software). This mode control switching is orthogonal to processor mode switching as seen in figure 4.

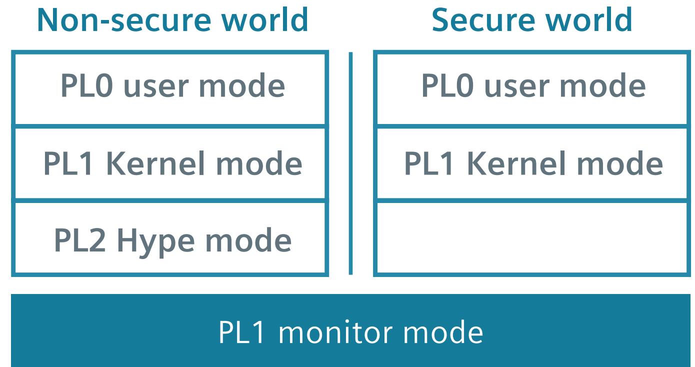

Figure 4. Current Armv7 CPU modes.

Although primarily designed for security, the TrustZone can be utilized as a method for hardware-assisted virtualization for mixed-critical systems. The TrustZone extension alone cannot be used to handle hypervisor code because there is no way to trap instructions for non-secure world to secure world which makes it impossible to virtualize different guests in the non-secure world. However this can be achieved using new HYP mode which reduces the complexity of hypervisor design and the cost of sensitive instruction emulation as it applies the trap- and-emulate technique using its own dedicated registers.

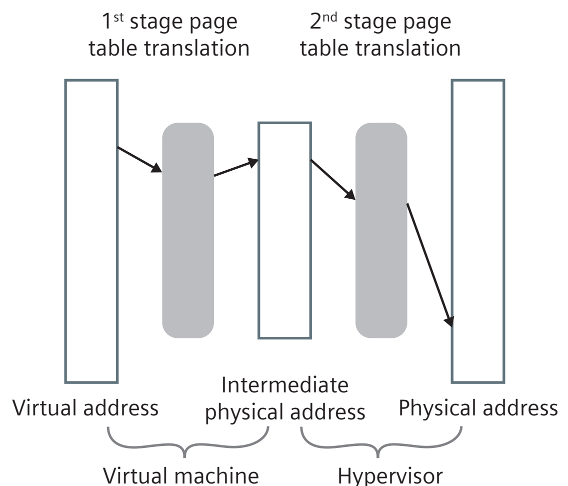

Figure 5. Memory addresses translation.

Memory virtualization

A new stage of address translation is added to decouple the management of memory addresses by the guest kernel from physical memory. This is achieved through Intermediate Physical Addresses (IPA) which are translated to Physical Addresses (PA) using 2nd stage page table translation. This stage is completely transparent to the guest kernel and protects physical memory from unauthorized access from guests. Figure 5 shows the stage of address translation for virtual machines running over a hypervisor.

It's also possible to remove 2nd stage address translation from HYP mode so that IPA and PA addresses are the same.

Arm's generic interrupt controller

The Arm generic interrupt controller consists primarily of two components:

- **The Distributor (GICD):** Performs interrupts prioritization and routing to all CPUs. Also responsible for software interrupts generation.
- **The CPU Interface (GICC):** Responsible for handling interrupts on the CPU level as it accepts interrupts from the distributor depending on the current priority level of the CPU. Responsible for acknowledging and signaling the end of interrupts.

Arm's virtual CPU interface

The virtualization extension for generic interrupt controller in Arm adds another component, which is the virtual CPU interface. The virtual CPU interface registers have the same programming model as physical CPU interface registers so the guest kernel will not be aware if it is communicating with the physical or virtual CPU. It will always use the addresses of the physical interface, but in case the software is virtualized, the hypervisor could use 2nd stage address translation to redirect that interface to virtual CPU registers. This design eliminates the need for emulating the CPU interface access so the guest OS can perform frequent tasks like interrupt acknowledgment more efficiently.

The hypervisor manages all physical interrupts through the distributor and routes them to the guest as virtual interrupts through list registers, which is a list that the hypervisor uses to maintain the state of virtual interrupts. In this way, the hypervisor virtualizes the functionality of the distributor for all guests, but this also means that guest access for distributor registers must be emulated.

In the new architecture of Arm's generic interrupt controller, the process of interrupt deactivation can be separated into two steps: 1) by lowering CPU priority, and 2) interrupt deactivation. Separating these two steps can be beneficial when doing virtualization. After the hypervisor receives a physical interrupt and routes it to the guest kernel as a virtual interrupt, it can lower the priority of the CPU while the virtual interrupt is still being processed. When CPU priority is lowered, new interrupts can be triggered which allows the hypervisor to prioritize received interrupts more efficiently before triggering them to guests. In this configuration, when the guest kernel deactivates the virtual interrupt, the physical interrupt will also be deactivated.

Generic timer

The generic timer module provides for each CPU a real-time counter and a timer that can be used to generate interrupts after configured periods of time. Typically, any kernel needs to have control and be able to manipulate a timer in order to schedule events in real time. This means in the case of virtualization, the hypervisor would have to emulate all access points of the guests to timers which is extremely inefficient. This would reduce real-time performance drastically. This is why generic timers in Arm architectures provide virtual counters and virtual timers, which can be used by the guest kernel without trapping the hypervisor. The guest kernel should be able configure the stop/restart virtual timers without hypervisor intervention, while the hypervisor uses physical timers directly for its own scheduling purpose. The virtual counter can be configured with an offset from physical counter so that each guest will have its own relative time. Moreover, virtual counters will automatically stop in case of switching to HYP mode which isolates the guests from hypervisor operation.

Siemens Embedded Hypervisor

There are two types of hypervisors commonly in use today (figure 6):

- **Type 1 Native Hypervisor:** A hypervisor that runs natively on hardware as it acts as an operating system in the core.
- **Type 2 Hosted Hypervisor:** This type of hypervisor must be hosted by another operating system, and is only responsible for virtualizing the guest operating system using the resources available to it from the host operating system.

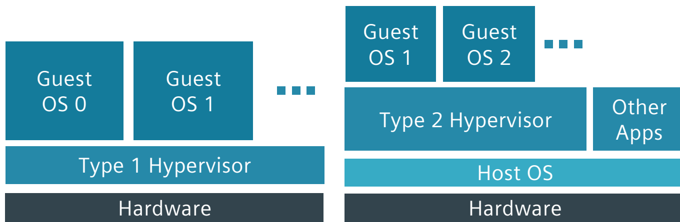

Figure 6. Interfaces across the XIL-levels do not match when data, languages, or tools change.

Siemens Embedded Hypervisor is a type 1 hypervisor which is more convenient by nature for embedded system applications where operating systems are simple enough to run natively on hardware with deterministic behavior. For Arm targets, the Siemens Embedded Hypervisor utilizes Arm virtualization extensions making the choice of simple type 1 hypervisor more appropriate in the design of CPU virtualization. Arm has introduced a new HYP mode which is not rich with registers like other modes so the hypervisor needs to be simple enough to utilize these registers. Siemens Embedded Hypervisor is targeted for embedded applications and is designed using microkernel architecture where the core of the kernel consolidates the most basic functionalities needed by the hypervisor to run on the physical hardware.

While the addition of other components and services is configurable, this approach has multiple benefits as it decreases the size of the trusted computing base in the system, making it easier to qualify the most critical parts of the system to the highest levels of safety. It also minimizes the memory footprint of the hypervisor according to guest needs such as unused functions which can then be configured to be removed from hypervisor image.

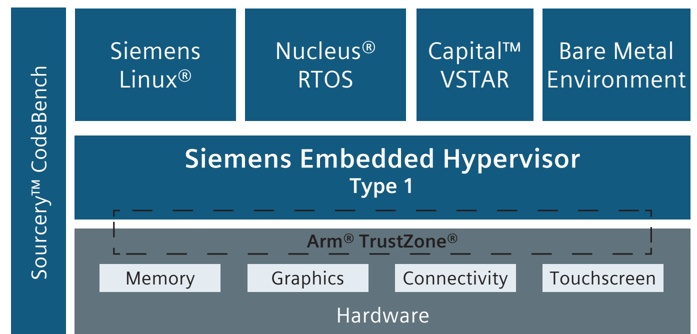

Figure 7. Siemens Embedded Hypervisor.

Siemens Embedded Hypervisor (figure 7) also utilizes Arm TrustZone to provide better isolation between virtual machines and supports multiple types of guest OSes such as:

- Android
- Capital VSTAR
- Siemens Embedded (Flex or Moni OS) Linux
- Nucleus RTOS

Hardware device access across guests

Siemens Embedded Hypervisor provides multiple models for hardware device access, allowing more flexibility in system design. Some of these models include:

- **Direct access:** Assigns hardware devices to be owned exclusively by a virtual machine which allows direct access. This is beneficial in case a hardware device is designed on the system level to be used by only one virtual machine as there will be no need for hypervisor intervention to improve system performance.
- **Shared access:** When a hardware device is owned by a virtual machine and realizes that this device is shared, so the handling is done on the virtual machine level.

- **Emulated access:** A common model for sharing hardware devices across virtual machines where the hypervisor controls device access through trap-and-emulate techniques while each virtual machine thinks that it owns the hardware device.
- **Virtual access:** Similar to emulated access where the hypervisor owns the hardware device, but the virtual machine realizes that it doesn't own the hardware device and uses a virtual driver for hardware access, thus eliminating the need for emulating the device, which decreases emulation overhead.

VirtIO support for virtual devices

The virtual device access model is considered a kind of para-virtualization technique as it requires virtual machines to implement a software interface with the hypervisor in order to handle virtual device access without the need for any kind of emulation.

VirtIO is a virtualization standard for para-virtualized device drivers, providing a standard application programmable interface so that it can be used by hypervisors and virtual machines to interact with common virtual devices. As seen in figure 8, a VirtIO architecture design consists of front-end drivers which are implemented in guests, back-end drivers which are implemented in the hypervisor, and virtual queues that handle communication between guest and hypervisor. Each VirtIO device will have its own front- and end-driver implementation.

The Siemens Embedded Hypervisor uses the memory mapped input output (MMIO) method for VirtIO devices and supports VirtIO Net, VirtIO Block and VirtIO Console devices. VirtIO is also supported in Linux.

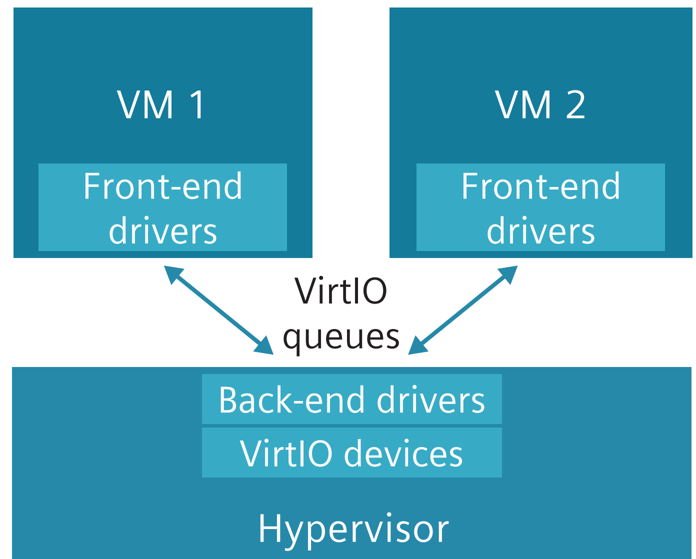

Figure 8. VirtIO architecture overview.

Virtualization for automotive software

Current approaches used to tackle the complexities described earlier in this paper (cockpit domain units) are both cost prohibitive and lacking in performance. Utilizing virtualization in automotive software architecture provides a better approach when taking on these complexities. This can be achieved by encapsulating different heterogeneous automotive platforms inside virtual machines running on the same hardware. This approach not only provides a more efficient way of communication by using inter-VM communication instead of serial connections, but also reduces the cost of adding a dedicated microcontroller to each platform. It also reduces the development cost by reusing legacy platforms as encapsulated virtual machines without the need for new adaptation efforts.

Inter-VM communication can be achieved using simple shared memory access techniques or by using more structured frameworks like VirtIO. Although AUTOSAR doesn't support VirtIO, it provides a specification named Complex Device Drives (CDDs) which allows integrating non-supported drivers like VirtIO in a standardized way.

Despite the benefits of virtualization for automotive software, the applicability of such an approach depends on the evaluation of application's performance - while running in a virtualized environment making sure the system's hard real-time requirements are still being met.

System implementation

This section describes how to encapsulate AUTOSAR and Linux guests in the Siemens Embedded Hypervisor to run on a TI Jacinto 6 infotainment platform.

Siemens Embedded Hypervisor configuration

Because the Siemens Embedded Hypervisor is designed as a microkernel, it allows the user to configure which drivers and components are added to the hypervisor image according to the guest needs. Guest information is supplied to the hypervisor through the device tree source files (DTS). These files are defined by the Siemens Embedded Hypervisor data driven configuration binding and is used by the hypervisor to know the number of virtual machines, the address mapping, the resources assigned to each virtual machine, and the number of virtual devices. After the Siemens Embedded Hypervisor parses the project configuration and generates the binary, it uses image tree source files (ITS) to package all of the guest binaries, along with the hypervisor binary, into a single monolithic image which is then deployable on the target hardware.

Virtualizing guests

For a Linux guest, a pre-built image is used for Siemens Embedded Linux, which is a commercial Linux distribution based on the Yocto® Project and includes a rich feature set useful for embedded applications.

Siemens Embedded Linux is virtualized through a para-virtualization layer which allows it to run over the hypervisor. For the AUTOSAR guest, Capital VSTAR implements the AUTOSAR standard. No virtualization efforts are needed for the VSTAR OS to run over Arm® Cortex-A15 hardware which shows the power of hardware-assisted virtualization extensions in Arm. But it's important to note what type of timers the OS supports. In the case of a virtualized environment, it would be impractical for the guest OS to use the physical timer directly as this would add emulation overhead that may affect the application performance. The alternative solution would be to either use the Arm virtual timer or to assign a dedicated general purpose timer for the virtual machine, so guest access to this timer will not need to be emulated.

Another key factor in guest configuration is a virtual machine to CPU core mapping. CPU sharing between multiple virtual machines introduces context switching overhead and increases the complexity of the design as it will depend on the scheduling algorithms used. This will impact application performance and will not be applicable to real-time virtual machines as deterministic behavior needs to be guaranteed in order to meet all hard timing requirements. For that reason, one-to-one mapping will be used between the virtual machines and CPUs.

AUTOSAR demonstration

For the AUTOSAR demo application, the goal is to achieve hard real-time requirements of the schedule table expiry points triggering. The key factors for this goal in the virtualized environment are:

- IRQ routing to virtual machine
- Generic interrupt controller access
- Virtual timer access

Figure 9 depicts an overview of this demonstration. The AUTOSAR application runs as VM1 on a dedicated core, the application utilizes the AUTOSAR cryptographic library to perform two operations which are MAC verification and AES decryption of encrypted data (every 100ms) and another application is tasked with

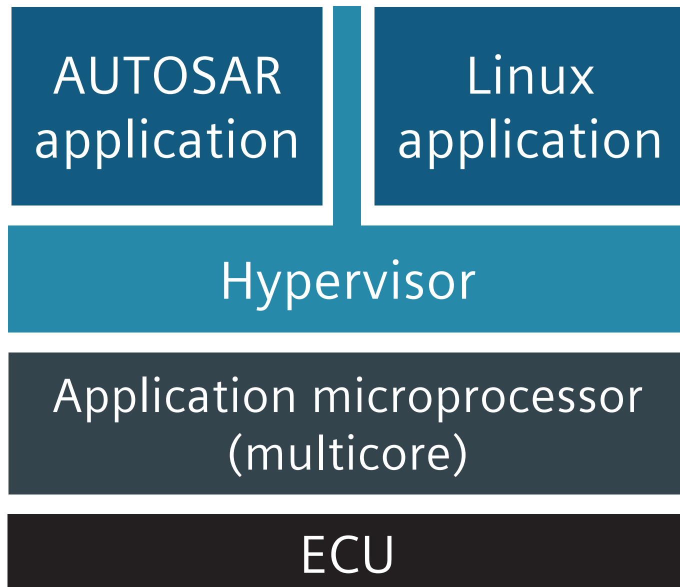

Figure 9. AUTOSAR demonstration overview.

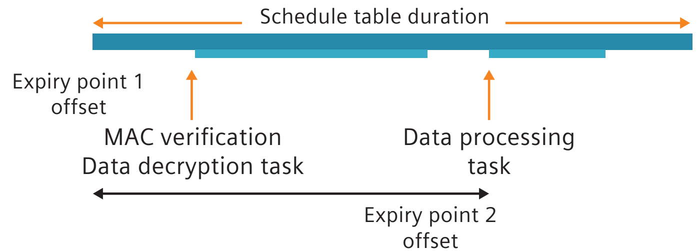

Figure 10. Schedule table and task overview.

processing decrypted data. This is organized using a schedule table (figure 10) which is an AUTOSAR OS entity that contains multiple expiry points with fixed offsets to assure relative synchronization between data decryption and data processing tasks.

The Linux application runs as VM2 on a dedicated core and shares UART console with the hypervisor to allow switching between hypervisor console and Linux console where the latter allows interfacing with the Linux file system (figure 11).

Figure 11. Switching between Siemens Embedded Hypervisor and Siemens Embedded Linux consoles.

AUTOSAR application runtime results

The total CPU load when running in virtualized environment increased only by ~0.2% which is the overhead added by IRQ routing and hypervisor scheduler handling. This represents the minimum overhead introduced by the hypervisor to run basic functionalities of the Capital VSTAR OS. The overall timing behavior of the AUTOSAR application is profiled to examine the effect of hypervisor overheads during application operation. This was done using a feature provided by the Capital VSTAR OS. This feature provides the ability to configure hooks at different points of the execution path like entering kernel, exiting kernel, changing the state of certain task, triggering of an event/alarm, and so on. These hooks can be used to monitor the timings of critical events during application runtime by reading the generic timer.

It was confirmed that using the AUTOSAR OS runtime measurement, the application performance was not affected when running in a virtualized environment.

Since there was no overhead for virtual timer access or GIC CPU interface access, kernel entry and exit times for the AUTOSAR OS didn't experience any changes when running in a virtualized environment. IRQ routing time was very short (561 nanoseconds for 1GHz clock frequency) so the latency for the OS timer interrupt was not affected. Moreover, the VSTAR AUTOSAR OS works as a tickless timer, where timer interrupts are only triggered on OS action points and not every fixed slice of time. As a result, the tickless timer feature limits the number of virtual timer interrupts needed and decreases the overhead of IRQ routing - schedule table expiry points and the time for scheduling tasks were not affected.

Conclusion

This paper has demonstrated how Siemens Embedded Hypervisor introduces a reliable solution for consolidating both infotainment and AUTOSAR applications on the same ECU. This is possible due to the efficient performance and the low cost associated with porting applications. Although this approach relies on hardware support, it is already commonly used today in infotainment domain ECUs.

The wide flexibility of the Siemens Embedded Hypervisor configuration allows the user to decrease the intervention of the hypervisor to minimal, which satisfies different application requirements even in situations where real-time applications with hard timing requirements are required.

Standardization of virtualization in automotive software architecture is an important step towards deploying advanced solutions in today's ECUs. For that very reason, Siemens Embedded Hypervisor supports standards such as VirtIO for virtual devices or AUTOSAR for in-vehicle, real-time applications. Developments such as these should encourage automotive standardization groups to adopt virtualization solutions in their standards.

Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. The Xcelerator™ portfolio, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

About the author

Mohamed Mounir is an embedded software engineer with hands-on experience working with AUTOSAR modules. He brings over 13 years of experience to his current role at Siemens. Mohamed has had the opportunity to work with a variety of tier ones and OEMs directly in different types of projects gaining wide experience in automotive software. Mohamed is currently receiving his M.Sc. degree from Ain Shams University where his research is focused on using virtualization concepts in automotive systems to achieve scalable and secure architectures.

siemens.com/software

© 2021 Siemens. A list of relevant Siemens trademarks can be found here.
The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks belong to their respective owners.

81433-C2 2/21 H