White Paper

Secure Virtualization for an Ever Increasing Vehicle Complexity

Visteon®

Table of Contents

Contributors:
Waheed Ahmed
Clemens Fischer
Gerald Harris
Harald Schoepp

Introduction 3

General Architecture: Virtualization Solution 4
- Classical Virtualization 4

Hardware Virtualization 5
- Background and ARM Virtualization Extension 5
  - Privileged Access Modes 5
  - Two-Tiered Memory Management Unit: Second Stage MMU 6
  - ARM Virtualization Extensions 6
- MOSS.x Virtualization: Philosophy 7
  - Dedicated Devices to Dedicated Cores 7
  - Securing Direct Memory Access (DMA) Capable Devices 7
  - Interrupt Distribution 7
  - Interrupt Based Inter-OS Communication 7
  - No Para-Virtualization 8
  - Protected Memory Regions for Virtualization Software and Kernel Images . 8
  - No Meta-Data from a Non-Secure Domain is Allowed in a Secure Domain . 8
- MOSS.x Virtualization-Software Architecture: System View 8
- Boot Process 9
- Memory Management 10
- Interrupt Handling 10
- Guest Reset 10
- Inter-VM Communication 10

MOSS.x Virtualization: Comparison with Classic Solutions 11

Conclusions 13

About Visteon 14

Introduction

The vehicle and mobility industry is dealing with the trends of ever-increasing complexity of electronic modules, software lines of code and consumer expectations. Today vehicles have up to 100 microprocessors on board, and that number is expected to double in the next five years. At the same time, the automotive vehicle has surpassed aerospace, consumer electronics and even the space shuttle in number of lines of software code, with over 100 million lines of code in some vehicles.

Led by the issues noted here, the automotive industry is following the trend of bringing different electronic domains onto a single platform. This leads to the challenge of enabling applications with diverse requirements i.e., security, safety and openness (open world) to work in a secure environment on a single platform. This encapsulation of different and diverse domains requires a novel virtualization solution. Visteon has developed such a virtualization solution, which in turn has become a pivotal component in realizing the fusion of disparate domains while fulfilling the security and safety requirements.

General Architecture: Virtualization Solution

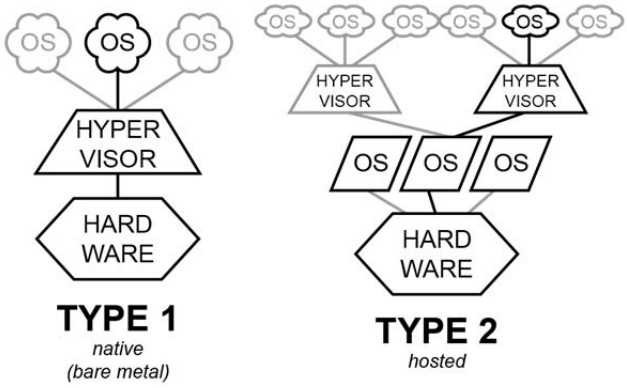
Classical Virtualization

To encapsulate different and diverse Operating System (OS) domains, a virtualization solution is required to ensure strict separation of the OS domains while enabling them to work together as components of a single system. This becomes challenging when the following domains are to be encapsulated on single system on chip (SoC):

- Cluster domain (safety critical), real-time constraints, avoiding denial of service.
- Cloud domain (requires open-world access).
- Entertainment domain (requires seamless and flexible user experience).

Virtualization can offer solutions for these problems. Quoting from Wikipedia[1], "Virtualization, in computing, refers to the act of creating a virtual (rather than actual) version of something, including but not limited to a virtual computer hardware platform, operating system (OS), storage device, or computer network resources."

"A hypervisor or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines."



"*Type 1* (or *native*, bare metal) hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems."

"*Type 2* (or *hosted*) hypervisors run within a conventional operating-system environment. With the hypervisor layer as a distinct second software level, guest operating-systems run at the third level above the hardware."

The concern with these classical virtualization solutions is the need to patch the operating system to fit within the virtualized environment. Patches are required to regulate access to shared physical resources and to ensure the best possible performance. This practice of patching the guest operating system is commonly referred to as "para-virtualization" and requires access to the source code of the guest OS. This is the reason why virtualization of proprietary operating systems (QNX, Windows) is not easily done.

Note that *Type 1* (or *native*, bare metal) hypervisors are also referred to as Hardware virtualization, while *Type 2* (or *hosted*) hypervisors are referred to as software virtualization.

[1] The following quotes and the image are taken from the Wikipedia articles for virtualization and hypervisors

Hardware Virtualization

Visteon has developed a virtualization solution, known as MOSS.x (multiple operating system security), based on full use of all available hardware support that modern processors offer. This MOSS.x virtualization is used in a line of domain controllers named SmartCore™. In comparison with available state-of-the-art virtualization solutions, we describe the MOSS.x virtualization as a Type 0 virtualization solution, eliminating the need for a software layer responsible for virtualization. Visteon's goal is to make maximum use of the virtualization support that the hardware offers. We strongly bind a guest OS to one or more cores of the target processor. This strong binding makes use of a number of hardware enforced security restrictions.

- Each core is only able to access a predefined area of physical memory
- Each core is only able to access a predefined subset of the peripherals
- Direct Memory Access (DMA) capable devices are subject to memory protection via a hardware feature available on SoC.
- All interrupts are delivered directly to the core that "owns" the interrupt initiator.

The strong binding results in an environment in which a guest OS can run on a specific subset of the cores of a multi-core processor without interfering with the operation of other guest OS's that are running on other cores. The guest will also run without any intervening software layer (traditionally termed as hypervisor); it will run directly on the underlying silicon. This enables full virtualization without the need to modify the guest OS.

Background and ARM Virtualization Extension

In order to realize the concept of hardware virtualization, a number of hardware extensions are required.

Privileged Access Modes

In a non-virtualized environment, two access modes are needed for normal operation:

- User level access
- Kernel access mode

Typically the user level access mode forbade direct hardware access as well as access to a number of the on-board processor registers. For example, it could have fatal consequences if an arbitrary user program were to modify the contents of the memory protection unit. This is an operation that is reserved for trusted code that resides in the OS kernel and which runs with kernel access privileges.

For virtualization, a third access mode is introduced: the HYP mode. This grants more and higher privilege levels than the normal kernel mode. In the context of a virtualized system, different OS's run on a single platform. Normally an OS will assume that it has unrestricted access to all resources of the underlying processor. Since this would have potentially fatal consequences in a virtualized environment a third mode is introduced for the virtualization software. The virtualization software is able to access the hardware platform under the highest privileged access mode.

Two-Tiered Memory Management Unit: Second Stage MMU

In a classic virtualized environment, each guest OS has its own address space. This requires an MMU to translate the logical address used by the guest applications to physical addresses. As the initial address generated by the MMU is an intermediate address, it introduces the need of a second stage MMU to translate the intermediate addresses from each guest to actual physical addresses. This ability of a second stage MMU to map the intermediate address to a physical address individually for each core provides two major benefits:

- It ensures that memory accesses from one guest cannot access physical memory that has been allocated to another guest.
- It allows memory mapped peripheral devices to be hidden from all guests except the “owner”. This is a new capability that is provided in modern SoCs.

The virtualization code retains ownership of the second stage MMU. It allocates physical memory areas to the different guests. The ability to define these maps statically at the time of configuration ensures that the intermediate to physical memory space mapping for every guest is defined so that they cannot violate each other’s memory space. The guest OS, in turn, has ownership of the first stage memory mapping, from the logical to the intermediate memory space.

This two-stage approach to memory mapping allows the guest OS the illusion of full ownership of memory mapping, as it would normally expect when running alone on the system, while allowing the virtualization-software control of an underlying layer of mapping to ensure that the different guests remain properly isolated from each other.

ARM Virtualization Extensions

In this section we summarize the main features of the virtualization extensions for the ARMv7-A architecture. These are new capabilities allowed under HYP mode and second stage MMU – as described by the ARM architecture reference manual – are:

- Separate HYP mode operating in its own virtual address space different from virtual address space accessed from PL0 and PL1 modes.
- Trap various operations including access to memory and coprocessor registers. Trapped operations generate an exception that is taken to HYP mode. Only the virtualization-software – running in HYP mode – can control these traps.
- Control routing of interrupts to guest or virtualization-software
- In PL0 and PL1 modes, MMU address translation occurs in two stages:
    - Stage 1 maps the Virtual Address (VA) to an Intermediate Physical Address (IPA). The guest operating system controls this stage, believing that the IPA is the Physical Address (PA).
    - Stage 2 maps the Intermediate Physical Address (IPA) to the Physical Address (PA). The virtualization-software controls this stage, and the guest operating system is completely unaware of this translation.

MOSS.x Virtualization: Philosophy
Based upon recent technical capabilities and market needs, the MOSS.x virtualization solution set was developed. Below we will outline the details of this solution and its unique aspects to deliver to the new market needs.

Dedicated Devices to Dedicated Cores
The SoCs that will commonly be used in high-end automotive infotainment applications will not only contain multiple processing cores. They will typically also contain multiple on-board peripherals. These might include device control ports (GPIO, I2C, SPI, UART), dedicated audio lines (TDM, I2S) or more complex controllers (Ethernet, USB, MOST). These peripherals will be addressable as part of a chip-internal address space.

Visteon's SmartCore™ virtualization solution ensures that each OS is being executed natively and therefore able to access Input and output (I/O) devices directly. This introduces the need to assign the I/O devices to individual cores (or OS's in our case). The ability of the second stage MMU to statically assign memory regions to particular cores allows the virtualization software to assign the memory-mapped I/0 and configuration regions to a guest as well. This assignment of I/O devices to cores is an important capability of ARM-based systems that we exploit to ensure that only the owner OS has access to its devices.

Securing Direct Memory Access (DMA) Capable Devices
The proposed virtualization solution controls the access for the cores (CPUs) to the rest of the system using the second stage MMU, but there are other hardware modules that can also affect physical memory, which are not protected by second stage MMU. These hardware modules use DMAs which operate directly on physical addresses.

DMAs are necessary, but can be dangerous for virtualization security due to their ability to access memory without any control.

We advocate using either the Memory-Firewalls across such devices or IP-MMU, where a second level of translation table specifies only the valid address translation. This approach is similar to second stage MMU, with the IP-MMU being used to control the physical memory accessible to DMA capable devices. We support both of these approaches and have implemented them using the latest features provided on TI-OMAP and Renesas SoCs.

Interrupt Distribution
Interrupts are usually associated with a specific hardware event and are therefore coupled to a specific peripheral device. It is possible to configure the processor to ensure that each interrupt is delivered to a desired core by configuring the generic interrupt controller (GIC) properly. Additionally we use the virtual GIC to direct the interrupts when traps are delivered to the virtualization software.

Interrupt Based Inter-OS Communication
The inter-guest (OS's) communication is based on shared memory and interrupts. We configure traps and write handlers to send interrupts between cores.

No Para-Virtualization
With full hardware virtualization we ensure security, isolation and portability. There is no need for para-virtualized drivers[2] for I/O access as each guest can access its dedicated peripherals directly.

Protected Memory Regions for Virtualization-Software and Kernel Images
We reserve certain portions of memory only for virtualization software code and kernel images. These areas are not accessible for any guest. We advocate loading the kernel images into this memory as backup images during the boot process. The virtualization code is invoked when a core "comes to rest" to allow it to reboot the individual OS's. In case of a crash of an individual guest OS, this property allows the rest of the system to continue to function while the crashed OS is rebooted.

No Meta-Data from a Non-Secure Domain is Allowed in a Secure Domain
The transfer of meta-data from non-secure guests to secure guest is not allowed as we don't expose any interface access (e.g. remote procedure calls) of secure guest.

MOSS.x Virtualization-Software Architecture: System View
Multiple Operating System Security (MOSS.x).

Figure 1 shows the overall system architecture of our virtualization solution which consist of MOSS.x Virtual Machine Manager (MOSS.x_VMM) and MOSS.x modules. The MOSS.x VMM is an implementation of Visteon's SmartCore™ virtualization solution. The MOSS.x modules consist of virtual devices that use shared memory to communicate between different guest operating systems.

Figure 1 illustrates how the MOSS.x VMM and MOSS.x modules interact with rest of system.

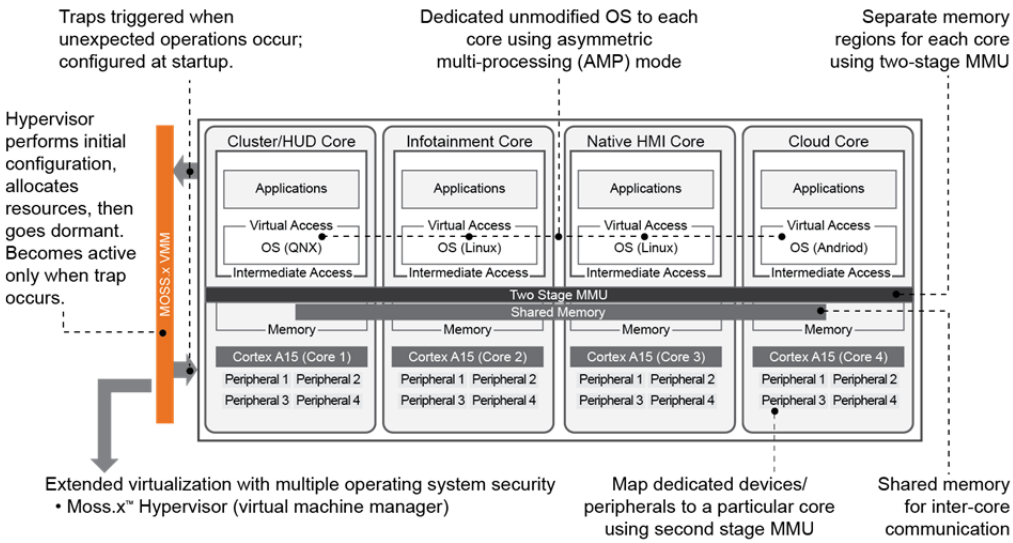


Figure 1. Overall Virtualization Architecture

[2] Para-virtualization refers to the practice of modifying the source code of a driver or of the guest OS itself to replace direct HW accesses with call to the hypervisor. Performance requirements make this essential in classical virtualization concepts.

Figure 2 shows the interaction between the MOSS.x VMM and the second stage MMU when implementing the allocation of address ranges to individual guests/cores. This determines both the allocation of physical memory and the allocation of memory mapped peripheral devices of the SoC.

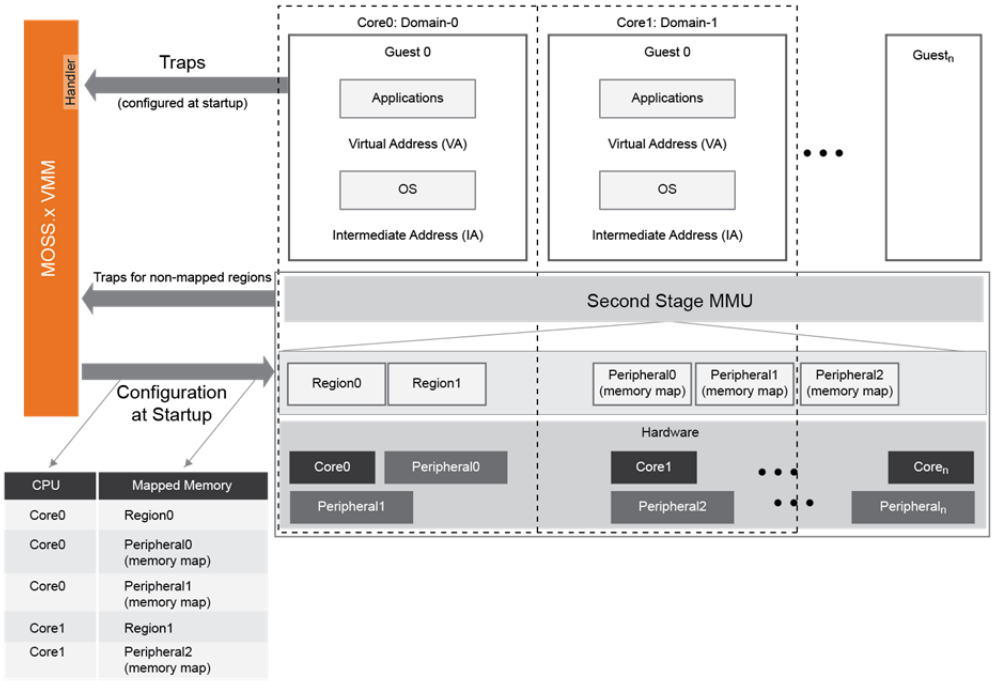


Figure 2. Use of second stage MMU

Boot Process

The boot process described here assumes an SoC with multiple cores and multiple guests.

The system firmware always starts the bootloader on core 0 in kernel[3] mode. The bootloader then loads the kernel images and device trees from the boot partition for all guests. The bootloader will then switch core 0 to HYP mode and will run the virtualization software. The virtualization will then proceed to boot the guest OS' on their cores with the boot of core 0 being done last. When a guest OS is started, the virtualization software will first initialize all required data structures. It will then start the required processor core (in HYP mode) to run a separate thread of virtualization software. Within this thread, several virtualization software registers will be initialized after which the core will switch to kernel mode and hands control over to the guest OS. After the guest OS on core 0 has been started, there is no active virtualization software running on the system – all that remains are the trap handlers that have been defined to deal with illegal memory accesses or to support communication between cores.

After boot-up there is no difference in the way in which different cores are handled. The trap handlers defined by the virtualization software will run on the core that triggered the trap.

[3] This is often referred to as "supervisor mode" in the literature which is not ARM specific.

Memory Management

An essential step in the virtualization software initialization process is the setup of page tables for the Stage 2 MMU. We chose to map each page 1:1, with matching physical and intermediate addresses. Note that the intermediate address space for each guest will only include the subset of physical memory that is intended for use by that guest.

When a guest attempts to access pages that are unmapped according to the Stage 2 page table, the processor raises an exception, causing it to switch to HYP mode and invoking the trap handler for data or instruction aborts.

To set up the page tables, we use the memory range information available in the device tree. This means that there is a single place to configure what devices are assigned to a guest and both the virtualization-software and the guest kernel can use that. For non-Linux guests, such as QNX, we still use the device tree in the virtualization-software, but it's not used by the guest kernel.

Interrupt Handling

As there is a 1:n mapping of OSs to cores and no dynamic scheduling is done by the virtualization software, the necessity of virtual interrupt management is reduced to a minimum. Only the assigned interrupts are delivered to the appropriate core and with the use of a virtual generic interrupt controller, they are directly delivered to the guest running on this core.

Guest Reset

Guests are not allowed to reset the whole system. However, there is still a need to support resetting of an individual guest, for example, to recover from error situations. Right before starting a guest, the virtualization software creates a backup copy of the guest kernel and device tree to a memory area which is owned by the virtualization software. When the guest attempts to reset the system, a trap will initiate a guest reset. This is implemented by restoring the guest kernel and device tree from the backup copy, reinitializing the state of the core, and then handing control back to the guest. By passing panic=-1 on the kernel command-line, Linux guests are instructed to issue an immediate reset whenever a kernel panic occurs.

Inter-VM Communication

A requirement for our virtualization software is to support communication between the two guests to enable sharing of frame buffers and transferring of key and touch events. We use shared memory and interrupts to accomplish this. With the memory management design described above, supporting shared memory is straightforward. When a particular physical memory range is specified in the device tree of both guests, it gets mapped to both cores and is accessible as shared memory.

For interrupts between guests we use the GIC to assert and clear otherwise unused interrupt lines. We added new property 'doorbells' to the device tree that describes what interrupts to trigger for communication with the other core. The doorbell is accessed using a trapped memory page, whose address is also described in the device tree. On the receiving end, the interrupt is cleared using the trapped memory page. This enables interrupt assertion and handling without any locking and with very low overhead compared with device interrupts.

MOSS.x Virtualization: Comparison with Classic Solutions

When comparing classical virtualization solutions with this new approach, several tradeoffs become apparent.

Classical virtualization approaches:

- Can be implemented on processors that do not provide the full array of hardware support features for virtualization. This includes just about every processor prior to the introduction of the A15 core from ARM.
- Allow creation of partitions (each with a guest OS) without worrying about the number of processing cores available.
- Support scheduling concepts that allow partitions with high CPU-time requirements to be dynamically assigned to cores when other partitions have lower requirements. This can result in more efficient usage of CPU time but means that the virtualization code is significantly more complex and then also becomes a consumer of CPU time.
- Require the ability to patch the OS (para-virtualization) and selected drivers to ensure the greatest possible efficiency. This in turn typically requires access to the source code of the OS, which makes virtualization a challenge for proprietary OS' such as QNX and Integrity.
- Are best recommended for non-safety critical applications, and/or applications with limited complexity.

Visteon's MOSS.x virtualization approach:

- Takes full advantage of hardware support for virtualization that was introduced with the A15 core.
- Restricts the number of partitions to be less than or equal to the number of available cores.
- Does no scheduling; the assignment of a partition to a (set of) core(s) is done statically during the boot process and is never changed while running. This maximizes the security between cores and domains.
- Takes full advantage of the ability to protect peripherals. A partition will never be able to access peripherals that have not been mapped into its address space.
- Requires no changes to the OS. Additional drivers may need to be added if different guests need the ability to communicate across partition boundaries.
- If processor flexibility is needed for non-safety critical features, then MOSS.x allows a subset of the available cores to be used in SMP mode under control of a single guest OS. This keeps the separation between AMP partitions, while allowing scheduling across multiple cores under control of the SMP guest.

When making the choice between solutions, this breaks down into a few simple considerations.

A classical virtualization solution will be required if one of the following conditions applies:

1) When using a processor which doesn't offer the needed hardware support for virtualization
2) When more partitions (guest OS') are required than the available processing cores on the processor.

**As noted above, best recommended for non-safety critical applications, and/or applications with limited complexity.*

However, if using classic virtualization it is important to note that access is required to the source code of the OS for purposes of para-virtualization. Using a proprietary OS will require significant support from the supplier, and they may want to dictate which virtualization supplier to use.

If both of these conditions apply, Visteon's MOSS.x solution can be used:

1) The processor is based on the A15 architecture or later with full virtualization support.
2) If para-virtualization is not an option, i.e., if a proprietary RTOS is required as one of the guests for which the source code is not available.

**Also note that the Visteon MOSS.x Type 0 virtualization solution provides increased security between domains and utilizes a minimum virtualization code size over Type 1 virtualizations.*

The MOSS.x solution in turn also provides the freedom to use any combination of OS' that the solution requires. No access to source code for purposes of para-virtualization is required.

Conclusions

Visteon has developed the MOSS.x SmartCore™ virtualization solution for state-of-the-art SoCs and is hardware supplier agnostic. This virtualization solution ensures the encapsulation of different and diverse domains in a fully virtualized environment while maintaining the native performance, security and safety requirements. The world's leading vehicle manufacturers are in need of a secure virtualization solution that allows for the flexibility of utilizing multiple OS's on the same SOC to allow for domain controller solutions to develop. This is the next evolution of the mobility system architecture and a viable way to implement all of the market driven changes needed to support the highly automated and autonomous future in mobility.

Visteon also provides classic virtualization solutions when required by the application. Analysis of customer system needs can be made through collaboration between OEM customers and Visteon Virtualization and Functional Safety teams.

About Visteon

Visteon is a global company that designs, engineers and manufactures innovative cockpit electronics products and connected car solutions for most of the world's major vehicle manufacturers. Visteon is a leading provider of driver information and controls, audio and infotainment, and domain controllers; its brands include Lightscape®, OpenAir® and SmartCore™. With corporate offices in Van Buren Township, Michigan, (U.S.); Shanghai, China; and Chelmsford, UK; Visteon has 50 facilities in 21 countries. Learn more at www.visteon.com.

Visteon Corporation
One Village Center Dr.
Van Buren Township, MI 48188
1-800-VISTEON
www.visteon.com

Copyright © 2015 Visteon Corporation