Designing Embedded Virtualized Intel® Architecture Platforms with the right Embedded Hypervisor

March 2011
Executive Summary

Virtualization is fast becoming a key enabling technology for embedded designs, offering the potential opportunity to consolidate multi-processor designs into a single processor multi-core design, legacy and proprietary embedded software migration and separation for performance and safety critical applications.

In this paper, we investigate the various hypervisor software models that exist and some of the design tradeoffs involved in order to help make a more informed decision when selecting hypervisor software for embedded virtualized designs based on Intel® Architecture.

As the use cases for embedded virtualization grow and the virtualization technology gets more pervasive in embedded multi-core systems, a critical design choice is which Hypervisor (virtualization enabling software) model will best fit the embedded application design needs.

Hypervisor software models offer different advantages and disadvantages based on:

- variation in software and hardware virtualization implementation strategy
- the kind of operating systems supported
- the inbuilt optimizations for certain application workloads and system security
These considerations make selecting an appropriate hypervisor model the number one design choice the architect needs to make while designing an embedded virtualized Intel® Architecture platform.

In this paper, we investigate the various software models that exist and some of the design tradeoffs involved in order to help make a more informed decision when selecting the hypervisor software for Embedded Virtualized Intel Architecture based designs.


Contents

Business Challenge ............................................................................................................. 5
Types of Hypervisor Software models ............................................................................. 5
Separation Kernel Model .................................................................................................. 6
Hypervisor Model .............................................................................................................. 8
Hybrid Hypervisor Model ................................................................................................. 10
Summary ........................................................................................................................... 11
References .......................................................................................................................... 12
Business Challenge

Hypervisor selection can be a tricky problem, especially when considering the design needs of an “embedded” virtualized platform. The challenge that is unique in the embedded space is how to maintain the deterministic behavior that characterizes an embedded application while hosting rich applications that may not demand a time sensitive response from the system. But the non-response-time-sensitive applications could be running alongside the response sensitive embedded applications and thus consume some of the processor, memory and other platform resources. Therefore, having virtualization software that recognizes this need in embedded platforms is critical for the successful implementation of virtualization technology in embedded designs.

The main task of any virtualization software stack is to allow multiple dissimilar operating systems to share a single physical hardware platform. However, the embedded virtualization software tries to solve the optimization problem of being able to provide the stringent real-time system requirements in terms of task scheduling and resources handling, while maintaining the richness of human machine interface typical for a general purpose operating environment. Some embedded virtualization software is better for real-time systems, some better for providing general purpose application hosting rich in graphics and some succeed in being able to provide an optimum balance.

The rest of this document explores some of this behavior and categorizes hypervisor software implementations based on what they try to achieve in an embedded virtualized environment.

Types of Hypervisor Software models

At a high level, there are three hypervisor software models that are commonly found in embedded Virtualization solutions: Separation Kernel model, Hypervisor model and Hybrid model.
A separation kernel (also called the safety or secure kernel) was first introduced as a means to achieve Multilevel Security or MLS policy employed in Military and Government systems (see [1] of the References section). The main idea of the MLS policy is to enforce an information security policy by classifying the security levels in a system as unclassified, confidential, top secret, secret, etc. The information access rights to individuals and groups within these security levels are then restricted and no individual or group can access information classified above their respective clearance level.

The basic premise that establishes the credibility of a Separation Kernel is:

- The System is broken down into a smaller and simpler trusted partition and a larger untrusted partition
- Security relevant decisions and operations are performed by the trusted portion making the untrusted portion irrelevant to the security of the overall system
- Trusted partition is passed through rigorous testing to ensure it possesses the necessary properties that constitute established security verification (see [2] and [3] of the References section)

So the basic philosophy behind the concept of separation kernel is to add complexity to reliability rather than attempting to add reliability to complexity the second being more common with the hypervisor model discussed in the next section.

If the separation kernel methodology is implemented well following the design/verification guidelines for security (see [2] and [3] of the References section) it will definitely provide some confidence in terms of security. But there are other factors like I/O design flexibility, legacy device emulation support, device sharing, system performance among others that need to be factored in when considering which model to use.
The separation kernel is a very thin software layer with code size typically between 5K to 10K lines capable of hosting operating systems similar to a native operating system hosting applications. Often times, it can also host critical applications directly on top of this kernel layer.

Separation kernel model software vendors claim that the main advantage to this model is the security aspect that this model brings in. The I/O devices on the embedded platform like USB, SATA, etc., can be assigned and dedicated to operating systems running on top, and these I/O devices can only write to the I/O buffers in the pin mapped memory allocated to these operating systems. This model leverages mainly from the Intel® Virtualization Technology for Directed I/O (Intel® VT-d) technology, which:

- enables the device assignment feature and
- uses DMA remapping engines in the chipset for memory read/writes from these devices to the pin mapped memory allocated to the operating systems associated with the these devices

For customers with embedded applications where information security is a critical element of the design, this is an attractive feature since there is no sharing and cross-interaction of I/O devices allowed.

As an example, a rogue application or system intrusion pretending to be IT support could try to install a software patch via a Windows admin console. This rogue application could not access or install new software on the virtual machine hosting the secure application through direct physical intrusion (via an external USB drive) or by working remotely on the system by trying to sneak in through the on-board Ethernet interface (assuming the system was architected to have separate NICs for each virtual machine).
However, it can be argued that this model may not be very flexible in terms of supporting physical device sharing or inter-VM/inter-process communication.

Some commercially available software solutions that may fit this model are Green Hills Integrity Secure Virtualization* and Lynuxworks LynxSecure* (see [8] and [9] of the References section).

**Hypervisor Model**

The Hypervisor model has two variants known commonly in the industry as Type1 and Type2 Hypervisors.

**Type 1 Hypervisor**

The first one is called Type 1 Hypervisor, the main distinguishing factor for this variant is that it is built from the ground-up for virtualization and hosting dissimilar guest operating systems. Much like the Separation kernel, this is a very thin layer also sometimes referred to as Bare Metal Hypervisor by some of the commercial Hypervisor software vendors.

![Figure 2 Different Layers typical for a Type 1 Hypervisor Model](image)

Some commercially available software that may fit this model are Real-Time-Systems Real-Time Hypervisor*, Wind River Hypervisor*, Tenasys eVM™ for Windows Embedded Virtual Machine Manager* (see [10], [11], and [16] of the References section).
Citrix Xenserver* is a commercially available virtualization solution based on Xen that follows the Type 1 Hypervisor model although it may not be as thin as some of the embedded hypervisors solutions from vendors like Real-Time-Systems or Tenasys. Another offering, the Citrix XenDesktop* brings forward an interesting business client virtualization usage model where complete Guest OS images and on-demand applications are dynamically hosted on client systems with XenDesktop installed on them. These images are actually running virtualized on Servers in the Data Center. The open source Hypervisor Xen hypervisor is also another example that has been widely studied in academia for understanding Virtualization and cloud computing. Xen is also a very competitive solution in the data centers for sever virtualization use cases. In terms of code size the Xen Hypervisor layer is claimed to be about 150,000 lines of code (see [4] and [19] of the References section).

The main advantage of a Type 1 Hypervisor is its lean memory footprint and ability to create separation between domains thus providing a fairly secure virtualization mechanism to host dissimilar application environments.

**Type 2 Hypervisor**

The second variant for the Hypervisor model is called the **Type 2 Hypervisor**, this kind of hypervisor is typically not built from the ground-up for virtualization, and it is actually an operating system modified to support virtualization. So the hypervisor layer which is a combination of what is called the Host operating system and virtualization software modules hosts guest operating systems. The hypervisor in this case uses the host operating system’s services and therefore the host OS has a bigger role to play in this kind of virtualized environment.

*Figure 3 Different Layers typical for a Type 2 Hypervisor Model*
A good example of this kind of model is KVM* (Kernel-based Virtual Machine) which is a free software released under GPL (see [17] of the References section). KVM is a full virtualization solution for Linux on x86 hardware containing virtualization hardware extensions (Intel® VT). It consists of a loadable kernel module kvm.ko that provides the core virtualization infrastructure and a processor specific module kvm-intel.ko. KVM also requires a modified QEMU. KVM is part of Linux and uses the regular Linux scheduler and memory management. KVM can only run on processors that support x86 hardware virtualization extensions. Unlike Xen, it does not support para-virtualization techniques for guest Oss but may support para-virtualization for device drivers to improve I/O performance.

Type 2 Hypervisors typically have a bigger memory footprint compared to the Type 1. These hypervisors are essentially an operating system modified so that they can host not just applications but complete operating systems. This can be advantageous in a scenario where you have a legacy operating system running a proprietary embedded application but you want to co-host a newer application that your legacy OS cannot support. One of the solutions in this scenario could be hosting a newer OS that supports the newer application on your current legacy OS environment. To accomplish this, you could use KVM (Type 2 Hypervisor) as an intermediary layer, which comes with added feature benefits that newer hypervisors like KVM bring.

This example can also work vice-versa, meaning hosting legacy environment on a newer OS with Type 2 Hypervisor installed.

**Hybrid Hypervisor Model**

One philosophy that distinguishes the hypervisor and separation kernel model is sharing of resources.
There is a third type of hypervisor model that takes the best out of both hypervisor and separation kernel which we will call the “Hybrid Hypervisor model”. Most hypervisor’s will have I/O device emulation built into this layer so that more than one Guest OS can access the on-board Network adapter as an example if needed. And if it is desired that the virtual machines can communicate with each other, then the easiest way is to share memory allocated to these virtual machines and some of the hypervisors. A lot of times a virtual network bridge is built-in for inter-virtual machine communication for the virtual machines being hosted by the hypervisor. These type of communication interfaces can be important for certain applications where one virtual machine supports data transceivers, real-time data acquisition and database management and then transfers this data via the communication interfaces to another virtual machine that supports data display, reporting, analysis and kick-starts another process based on the analysis performed. Industrial automation and control, medical devices, military Aeronautics and government are some embedded application segments that may have such use cases.

Hypervisor software solutions from Green Hills* and Lynuxworks* may fit this model and provide some of these hybrid hypervisor capabilities (see [8] and [9] of the References section).

**Summary**

There are various types of hypervisor software that are commercially available for embedded applications. They can be broadly classified into three software models: Separation Kernel, Hypervisor (Type1 and Type2) and Hybrid Hypervisor.
Designing an embedded virtualized platform with a hypervisor model that best meets the requirements of the end application is very important to meet the performance and reliability goals for real-time embedded systems.

Open source Type 2 Hypervisor Xen can be a good case study to understand the processor, memory and I/O device management virtualization techniques. These techniques and concepts should enable a system architect to make a more informed decision while deciding on which virtualization software to choose for implementing a quality embedded virtualized solution. For a detailed Case Study of Xen Hypervisor please refer to the Whitepaper titled “Xen* Hypervisor Case Study – Designing Embedded Virtualized Intel® Architecture Platforms” (see [20] of the References section).

References


Designing Embedded Virtualized Intel® Architecture Platforms with the Right Embedded Hypervisor


Authors

Amit Aneja is a Platform Architect with Embedded and Communications Group at Intel Corporation.

Acronyms

DMA Direct Memory Access
USB Universal Serial Bus
SATA Serial Advanced Technology Attachment
VM Virtual Machine
VMM Virtual Machine Monitor
HVM Hardware Virtual Machine