Xen* Hypervisor Case Study - Designing Embedded Virtualized Intel® Architecture Platforms

White Paper

Amit Aneja
Platform Architect
Intel Corporation

March 2011

Executive Summary

Virtualization technology is becoming popular beyond data centers. Corporate, Consumer and Embedded market segments are coming up with new use cases every few months, so virtualization enabling software companies need to quickly create new and innovative ways of utilizing hardware resources.

Virtualization enabling software companies need to quickly create new and innovative ways of utilizing hardware resources.

Embedded Virtualization enabling software models also have evolved in the past few years in order to cater to the stringent real-time and mission-critical demands of customers in the fast growing embedded systems industry.

In the white paper titled "Designing Embedded Virtualized Intel® Architecture Platforms with the right Embedded Hypervisor" (see [20] in the References section) we discussed the various hypervisor software models that exist to meet the embedded application design needs and the associated tradeoffs of these software models.

In this paper, we take the popular open source hypervisor, Xen (see [4] in the References section), for virtualization of x86 CPU architecture as a case study to get a deeper understanding of the various system design aspects when virtualizing an embedded system. This discussion should enable system architects to better understand how some of the design choices in embedded hypervisor software implementations may have been made and what could be the right design choice for the to-be-virtualized embedded application.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc.

Contents

Business Challenge

To meet the unique embedded application design needs, the embedded hypervisor software models implement various optimization techniques based on the embedded market segments they cater to. Also, while creating a virtualization enabled platform, the virtualization software developer must consider the overhead involved with changing the native configuration of one set operating system running on the hardware to multiple operating systems running on the hardware.

Intel® Virtualization technology provides hardware assists, new operating modes and enhancements to mitigate some of these virtualization enabling related issues. But these hardware assists and enhancements need to be implemented and carefully architected in the hypervisor software layer to reduce the overhead and increase overall performance and efficiency of the virtualization software.

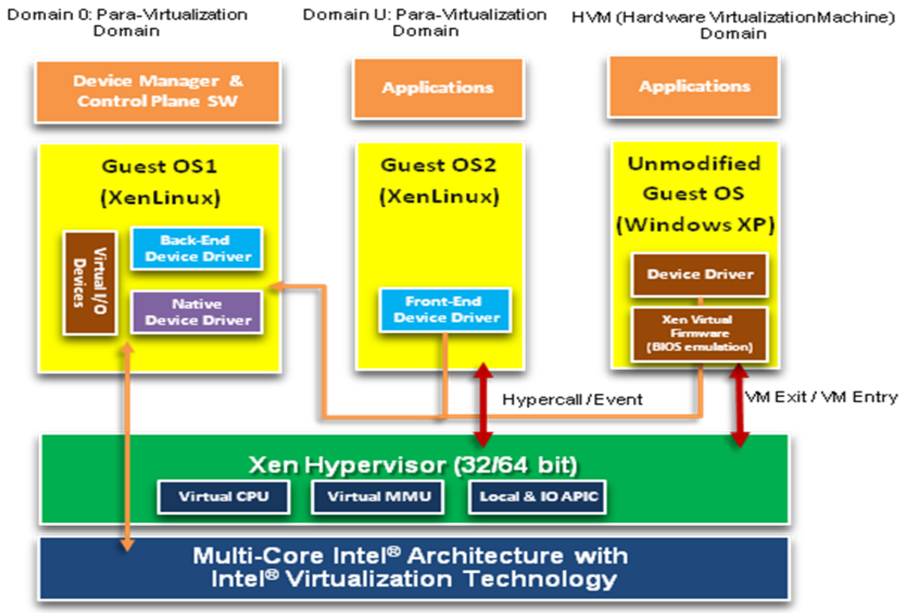
The processor, memory and I/O are the three most critical elements on any Intel® Architecture based embedded platform. Hypervisor models deploy techniques to better utilize the processor cycles by utilizing processor scheduling algorithms that best suit the embedded application needs, newer scheduling algorithms are being researched and implemented to more efficiently get the work done from the newer Intel® Hyper-threading technology enabled Multi-Core processors. Memory virtualization is another area of focus as the amount of main memory (DRAM) available can be the biggest factor in determining the number of virtual machines that a hypervisor allows to host on the system. Memory virtualization also determines how much memory is allocated to each virtual machine. Finally, I/O device virtualization can be important for applications that require a lot of I/O data traffic flow to-from and between virtual machines.

All the resource allocation, separation and isolation functions are typically performed by the hypervisor software. For this reason, it is important to explore and understand how a hypervisor performs some of these functions and the new virtualization techniques such as device pass through of I/O devices (also known as direct device assignment) and live migration. In the rest of this white paper, we discuss the open source Xen Hypervisor and see how some of these virtualization challenges are handled by Xen.

Xen Hypervisor – Case Study

There are a few questions that an architect should answer to determine the right Hypervisor software for the embedded application to be virtualized. We used Xen Hypervisor (see number [4] in References) as a case study to bring out some key aspects related to system architecture that will be applicable for most hypervisor software models and thus help in choosing the right virtualization solution.

Figure 1 Xen Hypervisor Architecture (Type1 Hypervisor)


Domain 0: Para-Virtualization Domain
Domain U: Para-Virtualization Domain
HVM (Hardware Virtualization Machine) Domain
Device Manager & Control Plane SW
Applications
Applications
Guest OS1 (XenLinux)
Virtual I/O Devices
Back-End Device Driver
Native Device Driver
Guest OS2 (XenLinux)
Front-End Device Driver
Unmodified Guest OS (Windows XP)
Device Driver
Xen Virtual Firmware (BIOS emulation)
Hypercall / Event
VM Exit / VM Entry
Xen Hypervisor (32/64 bit)
Virtual CPU
Virtual MMU
Local & IO APIC
Multi-Core Intel® Architecture with Intel® Virtualization Technology

Xen Hypervisor overview

Xen is a Type 1 Hypervisor, it contains three main components:

- Xen Hypervisor - The Xen hypervisor is the basic abstraction layer of software that sits directly on the hardware below the operating systems. It is a small hypervisor kernel that deals with virtualizing the CPU, memory and critical I/O resources such as the Interrupt controller.

- Domain0, the Privileged Domain (Dom0) – Dom0 is the Privileged guest (Para-virtualized Linux) running on the hypervisor with direct hardware access and guest management responsibilities. Dom0 is an integral part of the Xen-based virtualized system and is basically Linux OS modified to support virtualization. Dom0 also has a control panel application that controls and manages the sharing of the processor, memory, network and disk devices.

- Multiple DomainU, Unprivileged Domain Guests (DomU) – DomU's are the unprivileged guests running on the hypervisor; they typically do not have direct access to hardware (e.g., memory, disk, etc.). The DomU can be of two types – PV Guest (Para-virtualized Guest OS) and HVM (Hardware Virtualized Machine) Guest.

Dom0 and DomU operating systems use hypercall's to request services from the Xen hypervisor layer underneath. The HVM type DomU guests however use hardware based mechanism like VM Exit/VM Entry on Intel® VT enabled platforms to switch control between the hypervisor and HVM guest OS. The HVM guests are also known as fully-virtualized guests as the OS and device drivers run unmodified in their native configuration versus para-virtualized DomU guests where the guest OS or device driver code is typically modified to support virtualization.

Processor/CPU Management

The Hypervisor layer controls the most important task of optimizing the CPU utilization in a Virtualized environment. This function can be critical for overall performance particularly for system consolidation use cases. Some common tasks that the Virtual CPU module in the Hypervisor performs are CPU (socket/core) resource allocation and scheduling, VM execution control and VM migration services.

Xen allows a domain's/Guest OS's virtual CPU(s) to be associated with one or more host CPUs. This can be used to allocate real resources among one or more guests, or to make optimal use of processor resources when utilizing Intel Multi-core processors, Intel Hyper-Threading technology and other advanced processor technologies.

Xen enumerates physical CPUs in a 'depth first' fashion (see number [14] in References). For a system with both multiple cores and Intel Hyper-Threading technology, this would be all the threads on a given core, then all the cores on a given socket, and then all sockets. So, if you had a two socket, Dual Core, Intel processor with Hyper-Threading Technology, the CPU order would be:

Socket 0				Socket 1			
Core 0		Core 1		Core 0		Core 1	
Thread0	Thread1	Thread0	Thread1	Thread0	Thread1	Thread0	Thread1
VCPU#0	VCPU#1	VCPU#2	VCPU#3	VCPU#4	VCPU#5	VCPU#6	VCPU#7

CPU Scheduling

The credit CPU scheduler automatically load balances guest VCPUs across all available physical CPUs on an SMP host. The user need not manually pin VCPUs to load balance the system. However, the user can restrict which CPUs a particular VCPU may run on using the *xm vcpu-pin* command.

Each guest domain is assigned a weight and a cap. A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256. The cap optionally fixes the maximum amount of CPU a guest will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU, 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc. The default, 0, means there is no upper cap.

H/W Virtualization using Intel® VT-x technology

For the HVM domain, the Virtual CPU module in the hypervisor provides the abstraction of a processor to the guest. The Intel® VT-x technology provides VMX Root and Non Root modes of operation to allow running unmodified guest operating systems in HVM domain. Also, a VMCS structure is created for each CPU in the HVM domain. The VMCS data structure saves the guest and host states when transitions known as VM Exit and VM Entry happens between the guest domain and the hypervisor. Some of the VMCS control fields control the instructions and events that should trigger a VM Exit to the Hypervisor.

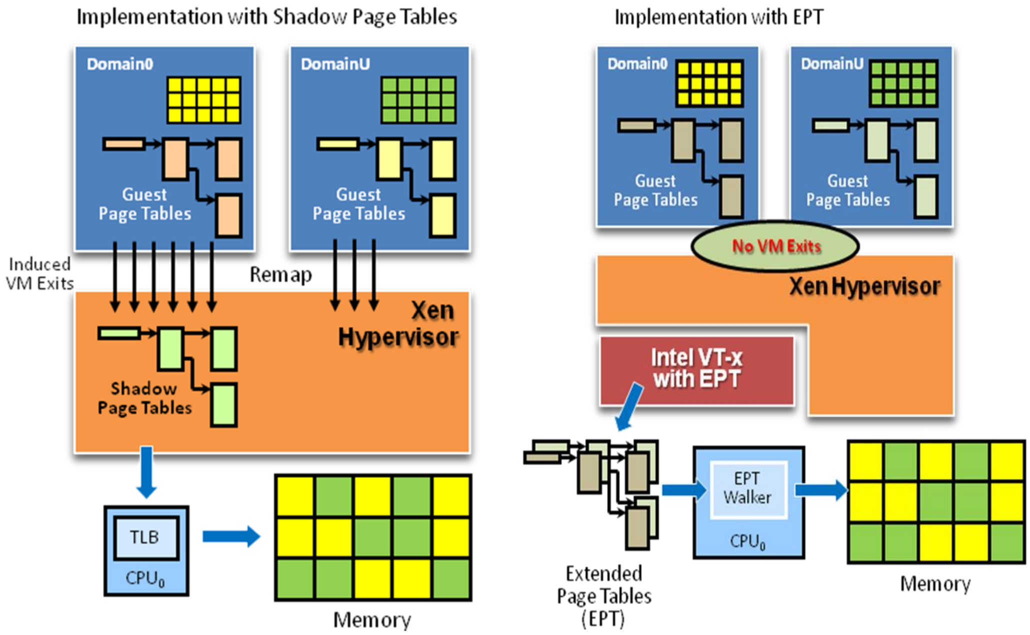
Memory Management

The Virtual Memory Management Unit (MMU) module in the hypervisor provides an abstraction of the hardware MMU to the guest operating systems. The main responsibility of the VMMU is to translate the guest physical address accesses to the appropriate machine physical addresses in the system memory.

Shadow Page Tables

For the guest domains running on the system, the Virtual MMU module maintains shadow page tables for the guests. The shadow page tables are the actual page tables the processor uses for mapping virtual address space to the physical address space in the system memory. Every time the guest modifies its page mapping, either by changing the content of a translation or creating a new translation, the virtual MMU module will capture the modification and adjust the shadow page tables accordingly. These operations can induce VM Exits (Non-Root to Root mode change/guest to hypervisor transition for HVM domains) every time there is a shadow page table access resulting in overall performance degradation (VM Exits add to overall execution time), besides the fact the shadow page tables consume significant memory space.

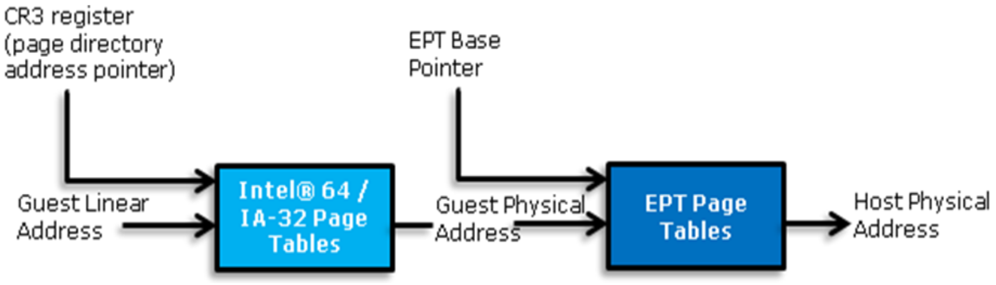
Figure 2 Implementation of Guest Paging mechanism



Extended Page Tables (Intel® VT-x)

The shadow page table code implementation can be critical from an overall performance point of view. To mitigate some of the page table shadowing related overheads, the concept of Extended page tables (EPT) was introduced in Intel Architecture based processors with Intel Virtualization technology enabled. EPT is optionally enabled in Xen when the guest is launched, when EPT is active the EPT based pointer loaded from the VMCS data structure (part of Intel VT-x technology) points to extended page tables. This mechanism allows the Guest OSs full control of the Intel 64/IA-32 page tables which map guest-linear to guest-physical addresses as shown in Figure 3.0. The new EPT page tables under VMM control and referenced by the EPT base pointer map guest-physical to host-physical addresses to access the memory allocated to the guest domains. This HW based mechanism prevents VM exits due to page faults, INVLPG, or CR3 accesses resulting in reduced paging related overheads.

Figure 3 Extended Page Tables on Intel Architecture



While efficient techniques for optimizing CPU and I/O device utilization are widely implemented, time-sharing of physical memory is much more difficult. As a result, physical memory is increasingly becoming a bottleneck in virtualized systems, limiting the efficient consolidation of virtual machines onto physical machines.

Transcendent Memory Concept

More recently advanced memory management technologies have been added to Xen. As an example Transcendent Memory (*tmem* for short) provides a new approach for improving the utilization of physical memory in a virtualized environment by claiming underutilized memory in a system and making it available where it is most needed (refer to Xen 4.0 release notes). More formally, Transcendent Memory is:

- a collection of idle physical memory in a system
- an API for providing indirect access to that memory

A *tmem host* (such as a hypervisor in a virtualized system) maintains and manages one or more *tmem pools* of physical memory. One or more *tmem clients* (such as a guest OS in a virtualized system) can access this memory only indirectly via a well-defined *tmem API* which imposes a carefully-crafted set of rules and restrictions. Through proper use of the tmem API, a tmem client may utilize a *tmem* pool as an extension to its memory, thus reducing disk I/O and improving performance. As an added bonus, tmem is compatible and complementary and also supplementary to other physical memory management mechanisms such as automated ballooning and page-sharing.

One of the limitations on tmem is overhead, which may result in a small negative performance impact on some workloads. OS change or para-virtualization may be required, though the changes are small and non-invasive. And *tmem* may not help at all if all of main memory is truly in use (i.e., the sum of the working set of all active virtual machines exceeds the size of physical memory) but overall the benefits of *tmem* can greatly outweigh these costs.

Guest Memory Page sharing

Another advanced memory management technique involves sharing memory space between virtual machines. The main purpose for sharing memory pages comes from the fact the identical guest OSs share a lot of identical data including common program binaries, shared libraries and configuration files. Sharing memory pages frees up additional memory for the hypervisor, reduces overall paging operations, increase page cache size and reduced I/O. There is some overhead related to memory scanning and finding identical pages to be shared, which may need additional research, but this technique helps improve overall performance and reduction in memory consumption by VMs. Xen 4.0 implements this concept of Memory page sharing and page-to-disc for HVM DomainU guests which means copy-on-write sharing of identical memory pages between VMs. This is an initial implementation and will be improved in upcoming releases.

I/O Device Management

Domain0, a modified Linux kernel, is a unique virtual machine running on the Xen hypervisor that has special rights to access physical I/O resources as well as interact with the other virtual machines (DomainU: PV and HVM Guests) running on the system.

A DomainU PV Guest contains para-virtualized front end drivers to support I/O device access through the Back end drivers in Domain0. An event channel exists between Domain0 and the DomainU PV Guest that allows them to communicate via asynchronous inter-domain interrupts in the Xen Hypervisor. The DomainU PV Guest virtual machine is aware that it does not have direct access to the hardware and recognizes that other virtual machines are running on the same machine.

Xen presents to each HVM guest a virtualized platform complete with an abstraction of keyboard, mouse, graphics display, real-time clock, 8259 programmable interrupt controller, CMOS, disk drive, CD-ROM etc, this support for the HVM guests is provided by the Virtual I/O Device module in Dom0.The control panel loads the Xen Virtual firmware into the HVM domain and creates a device model thread that runs in Dom0 to service I/O requests from the HVM guest. The control panel also configures the virtual devices seen by the HVM guest such as interrupt, DMA binding and the PCI configuration.

Other Key Design Considerations

Processor, memory and I/O device management are definitely the key design ingredients while designing a virtualized platform, but there are some other key features that need to be enabled as the virtualization use cases evolve. In this section we discuss some of these aspects using Xen Hypervisor as a case study.

Inter-process Communication

Many use cases of virtualization require a robust and fast inter-process communication to move data between virtual machines and the processes running in these separate domains. As an example, there could be embedded applications where data acquisition from the external environment using sensors will happen in real time fashion in one domain and then this data will be moved to a GUI based domain for analysis and system response. These scenarios will require a robust and reliable inter-process communication mechanism. In case of Xen, each domain network interface is connected to a virtual network interface in domain0 by a point to point link (effectively a “virtual crossover cable”). These devices are named vif<domid>.<vifid> (e.g., vif1.0 for the first interface in domain 1, vif3.1 for the second interface in domain 3).

Traffic on these virtual interfaces is handled in domain 0 using standard Linux mechanisms for bridging, routing, rate limiting, etc. Xend calls on two shell scripts to perform initial configuration of the network and configuration of new virtual interfaces.

By default, these scripts configure a single bridge for all the virtual interfaces. Arbitrary routing / bridging configurations can be configured by customizing the scripts.

PCI Device pass through (Intel® VT-d)

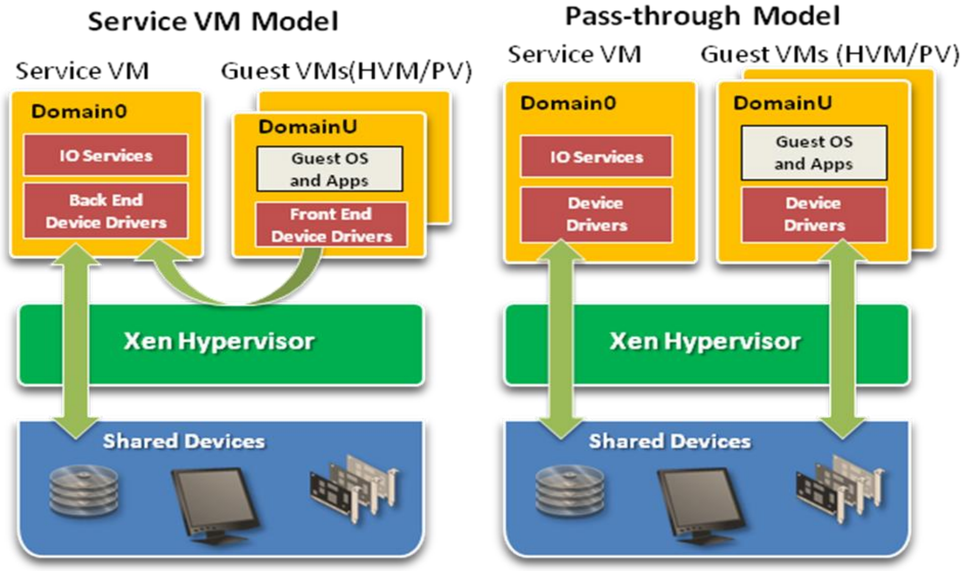
PCI device pass through is an important feature to enable direct device access from the Guest OSs which can be critical from the performance and device isolation perspective. In the case of Xen, Intel VT-d pass-through is a technique to give a DomainU exclusive access to a PCI function using the IOMMU provided by Intel VT-d. It is primarily targeted at HVM guests but PV (para-virtualized) guests can use similar pass-through technique.

Xen Hypervisor version 3.0 and greater supports PCI pass through model to assign a PCI device (NIC, disk controller, USB controller, Audio controller, etc) to a virtual machine guest, giving it full and direct access to the PCI device.

The guest VM (domainU) should have a driver for the actual PCI device, just like in the case of the PCI device on a native system (without Xen). Also, the Dom0 kernel must have pciback driver (xen-pciback in pvops dom0 kernels).

Xen supports this PCI device pass through model on both the para-virtualized (PV) guest OS and Hardware Virtualization Machine (HVM) guest OS. The enabling may however differ in these two cases.

Figure 4 I/O Virtualization models



The software based Xen PCI pass through to a PV domainU gives the guest full control of the PCI device, including DMA access (as shown in the Service VM Model side of Figure 4). This can be potentially unsecure and unstable, if the guest is malicious or has drivers with a lot of defects. Additional requirements for Xen PCI pass through to PV domU are PV domU kernel should have the Xen PCI front end driver loaded and some kernel options need to be added to the domU kernel cmdline.

In the case of Xen PCI pass-through for the HVM guest (as shown in the Pass-through Model side of Figure 4) the Xen HVM guest doesn't need to have a special kernel, or special PCI front-end drivers for the PCI pass-through to work. The main requirement is the correct configuration and enabling of hardware IOMMU (Intel VT-d) from the CPU/chipset/BIOS.

To enable IOMMU support at the first level, the BIOS option for IOMMU (Intel VT-d) should be enabled in the system BIOS. Some BIOS implementations call this feature "IO virtualization" or "Directed IO". If running Xen 3.4.x (or older version), iommu=1 flag needs to be added for Xen hypervisor (xen.gz) to grub.conf. Xen 4.0.0 and newer versions enable IOMMU support as a default if supported by the hardware and BIOS, no additional boot flags are required for the hypervisor.

"xm dmesg" Xen hypervisor boot messages can be read to check if "IO virtualization" gets enabled. For details on Xen PCI pass through usage and other features like Xen PCI pass through hotplug and hot-unplug, refer to [16] in References).

The more safe and secure hardware IOMMU (Intel VT-d) pass through model can also be used for PV guests, instead of the normal Xen PV PCI pass through. As a default Xen uses the normal (non-IOMMU) PV pass through for PV guests.

USB pass through

Xen supports device pass through for USB devices for HVM and PV guests using two different methods.

For Xen HVM guests (fully virtualized), the qemu-dm USB pass through using USB 1.1 emulation is used. Qemu-dm used for Xen HVM guests supports USB pass through of devices from dom0 to the guest. Qemu-dm emulates USB 1.1 UHCI 2-port controller, this method results in slower response time and is also limited in features and device support. It is however available in all Xen 3.x and newer versions and doesn't require any special drivers in dom0 kernel or in the HVM guest.

For Xen PV (para-virtualized) guests the PVUSB (Para Virtual USB) supported in Xen 4.0 and newer versions is used. PVUSB requires drivers in the kernel, in both the domainU kernel and domain0 kernel. PVUSB is a new high performance method of doing USB pass through from domain0 to the guests, supporting both USB 2.0 and USB 1.1 devices.

The new Para Virtual high-performance USB pass through method supporting USB 2.0 devices can actually be used with both HVM and PV guests. The guest kernel needs to have a PVUSB frontend driver installed. Also Xen dom0 kernel needs to have a PVUSB backend driver. Windows Xen PVUSB (Front end) drivers are still at an early version and will be likely added in newer release of Xen.

The Xen PCI pass-through technique can also be used to pass-through the whole USB controller (PCI device), including all USB devices connected to the USB controller if that meets the design USB device allocation requirements of the virtualized platform.

Pass-through of physical keyboard and mouse is also now possible using the Xen USB pass-through to pass through the USB keyboard and mouse devices or by using the PCI pass-through to pass through the whole USB controller managing the keyboard and mouse connected with it.

Graphics pass through

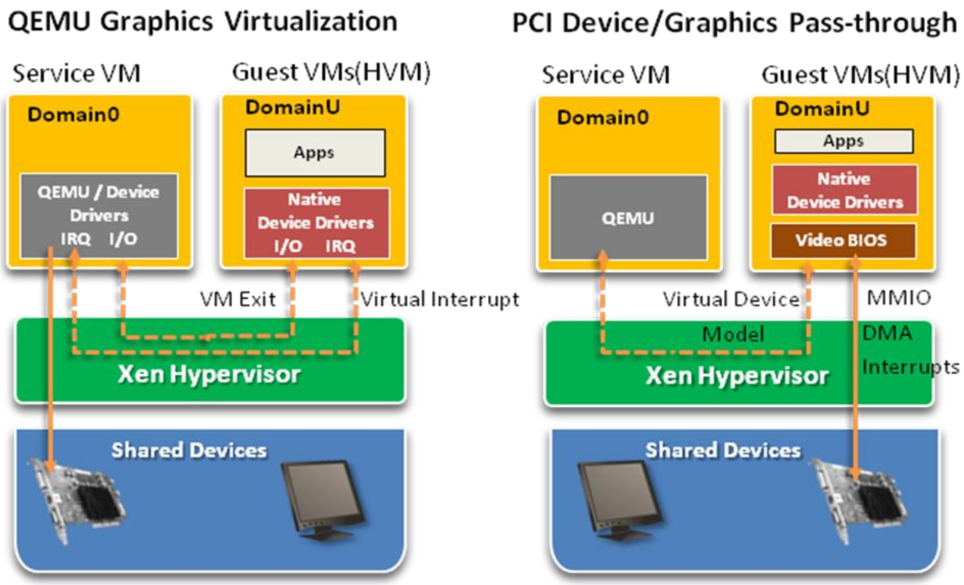
Xen version 4.0 now supports graphics adapter pass through to Xen HVM guests for high performance 3D graphics and hardware accelerated video enabling direct access to the processor graphics or the graphics card GPU from the guest OS. This means HVM guests can have full and direct control of the graphics engine of the system.

Xen graphics pass through requires IOMMU (Intel VT-d) support from the chipset, system BIOS, and ofcourse Xen.

Xen 4.0 however supports only graphics pass through of the primary graphics adapter (the one that gets initialized and is used when the system is powered-on). Graphics pass through of secondary graphics adapter currently requires additional patches to Xen.

Graphics adapters are not simple PCI devices like NICs or disk controllers. Graphics adapters need to support many legacy x86 features like Video BIOS, text mode, legacy IO ports, legacy memory ranges, VESA video modes, etc. for correct and expected operation. This means graphics pass through requires more code than normal Xen and Intel VT-d PCI pass through.

Figure 5 Graphics Virtualization in Xen



To provide a high level perspective of the enabling process, the qemu-dm emulator used in the Xen HVM guest needs to disable the internal (emulated) graphics adapter, copy and map the real graphics adapter Video BIOS to the virtual machine memory, emulate and execute it there to reset and initialize the graphics card properly, map and pass through the graphic adapters real memory to the HVM guest, pass through all the legacy IO-port and legacy memory ranges from the HVM guest to the real graphics adapter, etc.

At its current version 4.0, Xen graphics pass through is known to work with Intel Integrated Graphics Devices (IGDs) and with some primary graphics adapters which include Nvidia/ATI adapters (additional patches may be required to make them work).

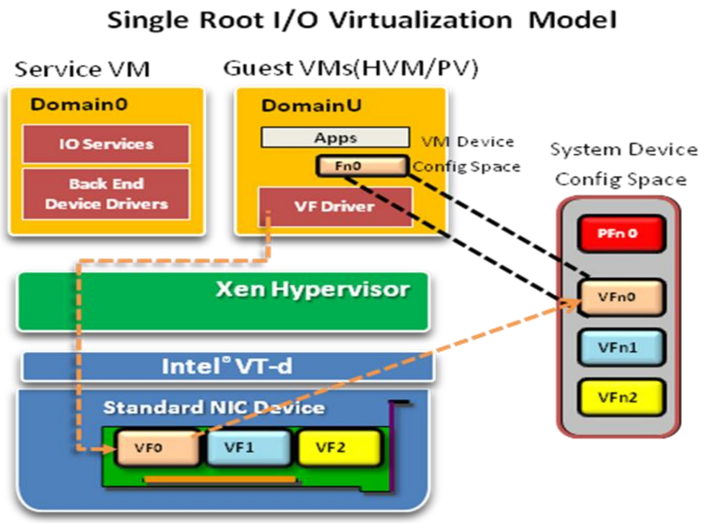
A lot of work to add more features is under development, like capability to pass through a second graphics card to the HVM guest. Patches also exist to support graphics pass through feature of Xen on Intel® Core™ and 2nd generation Intel® Core™ processor family based platforms.

SR IOV functionality

I/O Virtualization (IOV) involves sharing a single I/O resource between multiple virtual machines. Approaches for IOV include models where sharing is done completely in software, or where sharing is done in hardware, and hybrid approaches.

The direct device assignment method of I/O virtualization provides very fast I/O. However, it prevents the sharing of I/O devices. SR-IOV provides a mechanism by which a Single Root Function (for example a single Ethernet Port) can appear to be multiple separate physical devices.

Figure 6 VF Configuration mapping in SR-IOV model



A SR-IOV-capable device can be configured by the hypervisor to appear in the PCI configuration space as multiple functions, each with its own configuration space complete with Base Address Registers (BARs). The hypervisor assigns one or more VF to a virtual machine. Memory Translation technologies such as those in Intel® VT-x and Intel® VT-d provide hardware assisted techniques to allow direct DMA transfers to and from the VM, thus bypassing the software switch in the VMM.

Xen 4.0 supports the new advancements in networking hardware such as SMART NICs with multi-queue and SR-IOV functionality to provide virtualization infrastructure with improved data processing capabilities. Xen 4.0 takes full advantage of these new hardware technologies in the newly updated network channel feature called NetChannel2.

Live Migration

Migration is used to transfer a domain (virtual machine) between physical hosts. There are two common varieties: regular and live migration. The former moves a virtual machine from one host to another by pausing it, copying its memory contents, and then resuming it on the destination. The latter performs the same logical functionality but without needing to pause the domain for the duration.

In general when performing live migration the domain continues its usual activities and from the user's perspective the migration should be undetectable. In the case of Xen to perform a live migration, both hosts must be running Xen / xend and the destination host must have sufficient resources (e.g., memory capacity) to accommodate the domain after the move. Furthermore, current implementation requires both source and destination machines to be on the same L2 subnet. Currently, there is no support for providing automatic remote access to file systems stored on local disk when a domain is migrated. Administrators should choose an appropriate storage solution (i.e., SAN, NAS, etc.) to ensure that domain file systems are also available on their destination node. GNBD is a good method for exporting a volume from one machine to another. iSCSI can do a similar job, but is more complex to set up.

When a domain migrates, it's MAC and IP address move with it, thus it is only possible to migrate VMs within the same layer-2 network and IP subnet. If the destination node is on a different subnet, the administrator would need to manually configure a suitable etherip or IP tunnel in the domain 0 of the remote node. A domain may be migrated using the xm migrate command.

Remus – High availability/Fault Tolerance

New advanced technologies have been added to Xen to improve the fault tolerance and high availability of virtual machines. Remus Fault Tolerance is a new feature that provides live transactional synchronization of VM state between physical machines. Remus provides transparent, comprehensive high availability to ordinary virtual machines running on the Xen hypervisor. It does this by maintaining a completely up to date copy of a running VM on a backup server, which automatically activates if the primary VM fails. The backup is completely up to date. Even active TCP sessions are maintained without interruption. Protection is transparent and existing guests can be protected without modifying them in any way.

System requirements for running a hypervisor

When creating a virtualized environment, one of the main design considerations is the hardware that will be used on the system. A system architect needs to carefully select the processor, chipset and other silicon components that provide inbuilt hardware virtualization support. BIOS implementation is the next step where some of the hardware components and straps are initialized and enabled, so the BIOS implementation of virtualization features needs to be understood. Finally, Operating system support is a design consideration that is very important before choosing the hypervisor. It’s important to verify if the operating systems intended to be hosted on the virtualized platform are supported by the chosen hypervisor. When choosing a hypervisor, consider if it supports these technologies:

- general purpose operating systems like Windows and Linux
- commercial embedded and real-time operating systems
- built in to host proprietary operating systems which could be key for certain embedded systems.

Xen is a good example to understand what hardware and OS support may come into play.

Xen hypervisor 4.0 supports 32bit Physical Address Extensions (PAE)-capable CPUs and 64bit x86_64 CPUs. Also Intel IA-64 (Itanium) CPUs are supported.

CPU hardware virtualization extensions (Intel VT-x) are required for running Xen HVM guests (Windows etc). Xen PCI pass-through to HVM guests requires hardware IOMMU functionality (Intel VT-d) and support from CPU, BIOS and motherboard chipset.

Xen PV (para-virtualized) guests (Linux, Solaris, BSD, etc.) can be supported without CPU/hardware virtualization extensions. Xen 32bit PV guests need to be PAE.

Xen.org mentions that Xen 4.0 hypervisor has been tested the most on 64-bit CPUs, so it's recommended to build and run 64bit Xen 4.0 hypervisor. Domain0 kernel can still be 32 bit PAE even when the hypervisor is 64 bit (if you're not able to run full 64bit environment for both Xen+dom0). Xen guests (VMs) can be 32 bit when using 64 bit Xen hypervisor.

Summary

Understanding how the hypervisor layer controls resource allocation such as processor, memory and device I/O is very important to find the right embedded virtualization software solution. The Type 1 Hypervisor Xen is an evolving virtualization solution that implements many known resource control and optimization techniques and can be a good case study for understanding how hypervisor layer is architected in a virtualized solution. System architects can ask the commercial hypervisor software vendors how some of these virtualization techniques are implemented in their software solutions to get the right hypervisor software for their embedded application design needs.

References

[1] J.M. Rushby and B. Randell, "A Distributed Secure System", IEEE Computer, Volume 16, Issue 7, July 1983 Page(s):55 - 67

[2] J. Alves-Foss, P.W. Oman, C. Taylor and W.S. Harrison,"The MILS architecture for high-assurance embedded systems," *International Journal of Embedded Systems*, vol. 2,no. 3/4, pp.239-247, 2006.

[3] J. Alves-Foss, C. Taylor and P. Oman. "A Multi-Layered Approach to Security in High Assurance Systems," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences,* Waikola, HI, IEEE Computer Society, 2004.

[4] Xen Hypervisor is developed and maintained by the Xen.org community and available as a free solution under the GNU General Public License. For more information, see http://www.xen.org/

[5] A. Aneja, J. Achar "Introduction to Security using Intel® Virtualization Technology (Intel® VT) and Intel® Trusted Execution Technology(Intel® TXT)", Intel® Embedded Design Center http://edc.intel.com/Video-Player.aspx?id=2393

[6] A. Aneja, "Exploring how Intel® Virtualization Technology Enables Virtualization", Intel® Embedded Design Center http://edc.intel.com/Video-Player.aspx?id=3102

[7] J. St. Leger, "Virtualization: Technology basics and Business fundamentals", Intel® Embedded Design Center http://edc.intel.com/Video-Player.aspx?id=3101

[8] http://www.ghs.com/products/rtos/integrity_virtualization.html

[9] http://www.lynuxworks.com/virtualization/hypervisor.php

[10] http://www.real-time-systems.com/real-time_hypervisor

[11] http://www.windriver.com/products/hypervisor/

[12] http://www.redbend.com/index.php?option=com_content&view=article&id=134&Itemid=60&lang=en

[13] D. Neumann, D. Kulkarni, A. Kunze, G. Rogers, E. Verplanke, “Intel® Virtualization Technology in embedded and communications infrastructure applications”, Intel® Technology Journal, Volume 10, Issue 03, August 10, 2006

[14] http://bits.xensource.com/Xen/docs/user.pdf

[15] http://oss.oracle.com/projects/tmem/

[16] http://wiki.xen.org/xenwiki/XenPCIpassthrough

[17] http://www.tenasys.com/products/evm.php

[18] http://www.linux-kvm.org/page/Main_Page

[19] PCI SIG SR-IOV Primer, An Introduction to SR-IOV Technology, Intel® LAN Access Division, http://download.intel.com/design/network/applnots/321211.pdf

[20] A. Aneja, “Designing Embedded Virtualized Intel® Architecture Platforms with the right Embedded Hypervisor”, Intel® Embedded Design Center

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel’s tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc.

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

SMP Symmetric Multi-processing

GUI Graphical User Interface

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice.

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.