Hardware-assisted Virtualization with the MIPS® Virtualization Module

Document Number: MD00994
Revision 01.00
December 3, 2012

MIPS Technologies, Inc.
955 East Arques Avenue
Sunnyvale, CA 94085-4521

Copyright © 2012 Imagination Technologies Inc. and/or its affiliated group companies. All rights reserved.

Unpublished rights (if any) reserved under the copyright laws of the United States of America and other countries.

This document contains information that is proprietary to MIPS Technologies, Inc. ("MIPS Technologies") one of the Imagination Technologies Group plc companies Any copying, reproducing, modifying or use of this information (in whole or in part) that is not expressly permitted in writing by MIPS Technologies or an authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

Any document provided in source format (i.e., in a modifiable form such as in FrameMaker or Microsoft Word format) is subject to use and distribution restrictions that are independent of and supplemental to any and all confidentiality restrictions. UNDER NO CIRCUMSTANCES MAY A DOCUMENT PROVIDED IN SOURCE FORMAT BE DISTRIBUTED TO A THIRD PARTY IN SOURCE FORMAT WITHOUT THE EXPRESS WRITTEN PERMISSION OF MIPS TECHNOLOGIES, INC.

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS Technologies or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

MIPS, MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPSr3, MIPS32, MIPS64, microMIPS32, microMIPS64, MIPS-3D, MIPS16, MIPS16e, MIPS-Based, MIPSsim, MIPSpro, MIPS-VERIFIED, Aptiv logo, microMIPS logo, MIPS Technologies logo, MIPS-VERIFIED logo, 4K, 4Kc, 4Km, 4Kp, 4KE, 4KEc, 4KEm, 4KEp, 4KS, 4KSc, 4KSd, M4K, M14K, 5K, 5Kc, 5Kf, 24K, 24Kc, 24Kf, 24KE, 24KEc, 24KEf, 34K, 34Kc, 34Kf, 74K, 74Kc, 74Kf, 1004K, 1004Kc, 1004Kf, 1074K, 1074Kc, 1074Kf, R3000, R4000, R5000, Aptiv, ASMACRO, Atlas, "At the core of the user experience.", BusBridge, Bus Navigator, CLAM, CorExtend, CoreFPGA, CoreLV, EC, FPGA View, FS2, FS2 FIRST SILICON SOLUTIONS logo, FS2 NAVIGATOR, HyperDebug, HyperJTAG, IASim, iFlowtrace, interAptiv, JALGO, Logic Navigator, Malta, MDMX, MED, MGB, microAptiv, microMIPS, OCI, PDtrace, the Pipeline, proAptiv, Pro Series, SEAD, SEAD-2, SmartMIPS, SOC-it, System Navigator, and YAMON are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries. All other trademarks referred to herein are the property of their respective owners.

All other trademarks referred to herein are the property of their respective owners.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

1 Introducton

The primary purpose of this paper is to describe key features of MIPS Technologies' Virtualization solution, an optional module that is incorporated in the MIPS32® and MIPS64® architectures. In the course of the discussion, it will be made evident how a MIPS virtualization-compliant processor differentiates itself from existing solutions while retaining the simplicity, flexibility and completeness of the baseline MIPS architecture. In addition, this paper will summarize unique applications of hardware-assisted virtualization and describe how MIPS' solution can be used to address such applications in an efficient manner without significant performance impact that is commonly associated with virtualized solutions. A companion paper will describe in detail the role of virtualization in the area of application security.

1.1 Background on Virtualization

Virtualization is a technique used to enable sharing of common resources available on a hardware platform among logically separate virtual machines, where the virtual machine is comprised of an operating system's software stack and its applications. The common resources managed in this manner are the processor, memory, hardware accelerators, and I/O devices among others.
.
Software-based resource sharing is common. An Operating System (OS), as an example, abstracts all underlying hardware resources from a system's applications. This abstraction layer between application programs and resources allows an application to be written independently of the hardware platform, easing a programmer's task. In addition, the abstraction allows a new hardware platform to retain compatibility with legacy applications.

Virtualization requires another abstraction layer that sits between the OSes and the underlying hardware resources, which manages the virtual machines. This software layer is called the *Virtual Machine Monitor*, or *Hypervisor*. The operating systems are referred to as *guests*, because they are no longer directly interacting with hardware.

Virtualization as it is commonly understood today is typically *processor* virtualization, whereby the processor is considered a resource that needs to be shared among multiple virtual machines. Another example of virtualization in a System-On-Chip (SoC) implementation is *device virtualization* in the form of a Virtualized I/O Memory Management Unit, or IOMMU, whereby direct memory access (DMA) by a device can be virtualized. Hardware virtualization can also extend from the interface of the system to the network in the form of *network* virtualization. The I/O Virtualization (IOV) extensions to PCIE are an example of such support.

This paper focuses on *processor virtualization* with the MIPS Virtualization Module.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

1.2 Different types of processor virtualization

There are a few types of processor virtualization, namely *Para-virtualization* and *Hardware-assisted Virtualization*.

1.2.1 Para-virtualization

This form of virtualization requires modifications to the OS to support explicit intervention by a hypervisor. It is chiefly meant for legacy processors with no architectural support for virtualization. Para-virtualization is useful since there will always be processors that don't support hardware-assisted virtualization. For these chips, the related application may require virtualization for which a para-virtualized solution is adequate.

A potential drawback of such a scheme is that any OS compatible with the non-virtualized platform must be modified. This may limit the number of OSes that can be ported to the virtualized system. The modifications result in the OS being *deprivileged* i.e., it runs as an application. The modifications, primarily in the area of memory management and privileged access, may potentially impact performance, unless the hypervisor framework accounts for these issues.

For example, a Translation Lookaside Buffer (TLB) miss requires hypervisor intervention to reconstruct the *system physical address* for the guest OS. In this case, the physical address supplied by the OS is still *virtual*, though the OS is unaware of this fact. This requires, in addition to a walk of the page table of the guest OS, a walk of the hypervisor page table for the system physical memory allocated to the guest OS. The additional walk requires hypervisor intervention and may impact the overall latency of the walk.

Options exist to reduce and possibly eliminate such impacts. For example, the hypervisor may maintain a *shadow page table* for a guest OS that maps its virtual address to the system physical address, thus reducing the number of page table walks to one. However, maintaining a shadow page table also requires the hypervisor to monitor guest actions that modify the guest page tables so these actions can be replicated in the shadow page table. Further, since the shadow page table contains system physical addresses, only the hypervisor can walk the table so the penalty for a hypervisor call by the guest OS is not completely eliminated in such a scheme.

An optimal para-virtualized environment thus requires fine-tuning of the modified OS and hypervisor to avoid potential performance impacts. Such hypervisors and OS ports exist in the market, though there is an alternative to para-virtualization in the form of hardware-assisted virtualization.

1.2.2 Hardware-assisted Virtualization

The intent of hardware-assisted virtualization is to minimize software intervention on the part of hypervisor by moving more virtualization functionality to hardware. Hardware-assisted

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

virtualization does not require modification of a guest OS. As long as the OS is compatible with the underlying baseline architecture, it must be able to operate unmodified on the corresponding virtualized implementation. This requirement is a goal of any hardware-assisted virtualization architecture.

The fact that the guest OS does not have to be modified adds to the flexibility of hardware-assisted virtualization. Multiple virtual machines, each with a *different unmodified* OS, can run on a common processor, without regards to compatibility. Further, the OS is never deprivileged i.e., it operates with full *virtual* privilege but under the control of the hypervisor

As mentioned, one of the purposes of the hardware-assists is to optimize the performance of a virtualized solution. Taking the prior example of the TLB miss, a processor with hardware-assists will typically provide separate TLBs for the guests and hypervisor context, which are independently managed by the guest OS and hypervisor respectively. This eliminates the overhead of an additional page table walk for every guest TLB miss, and the need for the hypervisor to shadow the page table operations of the guest OS.

In addition to eliminating limitations of para-virtualization, hardware-assisted virtualization also allows for direct hypervisor control over a guest as a preemptive measure for avoiding unsafe guest behavior. Such control is available only to a limited extent with para-virtualization, since software by itself cannot be made aware of all such hardware events.

An additional benefit of hardware-assisted virtualization is that the hypervisor itself is simplified. A simplified hypervisor can be made *secure* through formal mathematical proofs, which allows such a hypervisor to act as a *Root of Trust* on a platform for secure applications such as encrypted media downloads, or those requiring mobile payments.

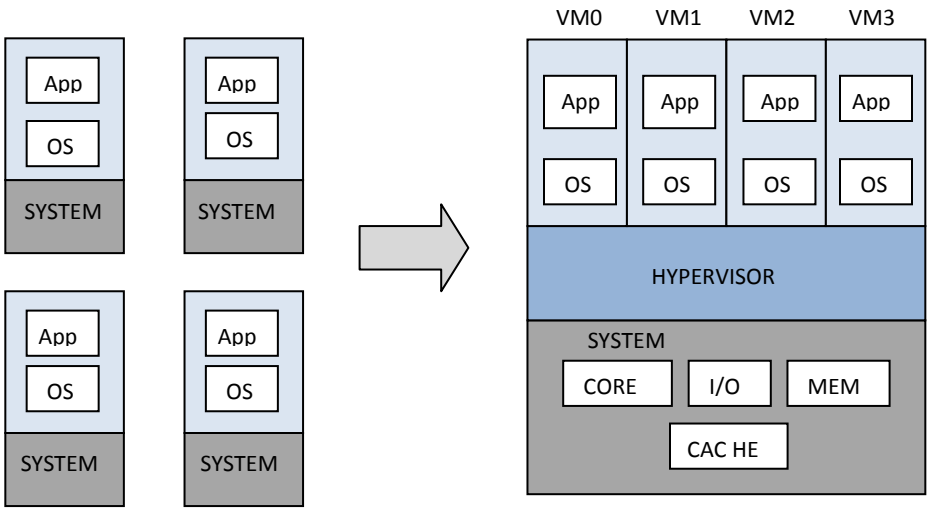

Figure 1: Non-Virtualized to Virtualized System Consolidation

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

2 MIPS Virtualization Module

The MIPS Virtualization Module is an optional module that is specified in the base MIPS32 and MIPS64 architectures to support hardware-assisted virtualization. It retains the original simplicity of the MIPS baseline architecture while providing a host of features in support of virtualization.

The salient features will be discussed one-by-one. To a reader who is familiar with various processor virtualization architectures, the advantages of the MIPS solution should be self-evident.

2.1 Terminology Specific to the MIPS Virtualization Module

The baseline MIPS architecture provides a set of privileged control and status registers called *Coprocessor0*, or *COP0* in short, to the OS. In a virtualized implementation, in addition to a COP0 for the guest OS, there is an instance of COP0 exclusive to the hypervisor. The COP0 state for the guest OS is the *Guest* context, while the COP0 state for the hypervisor is the *Root* context of the processor. References to guest and root in this document generically refer to software operating in their respective contexts.

Multiple guests can coexist in a processor through the optional use of *Guest Identification or GuestID*, which is a unique tag allocated by the hypervisor to each guest. This allows the hypervisor to switch contexts without having to invalidate the prior guest, or root context.

A transition from root to guest is called a *guest entry*. A transition from guest to root is called a *guest exit*.

2.2 Salient Features of the MIPS Virtualization Module

2.2.1 Separate Architectural Privileges for Root

The baseline architecture provides user, supervisor and kernel privilege levels to software. In the MIPS Virtualization Module, one such set of privilege levels is allocated to guest and another to root by providing each a copy of COP0. This allows guest software to run with full *virtual* privilege. This is in contrast to a para-virtualized solution where a guest's operation is deprivileged because it must run at the lower privilege of user while the hypervisor operates at kernel privilege.

Since guest software can run with full privilege, guest privileged instructions can execute without trapping to the hypervisor as long as the hypervisor enables such behavior through control provided by the Virtualization Module. Further, exceptions and interrupts may be serviced by a guest kernel, as long as they are enabled by root.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

Root privilege is always higher than guest privilege, and its application in a processor follows the *Onion* model. The Onion model can be understood through the handling of interrupts and exceptions. Interrupts are processed *outside in* – first by root interrupt rules, then by guest interrupt rules, if permitted. On the other hand, guest exceptions are processed *inside out* – first by guest exception rules, and if required, by root exception rules. The rules in either case are defined by the MIPS Virtualization Module.

The application of the Onion model allows for clean, unambiguous interaction between root and guest.

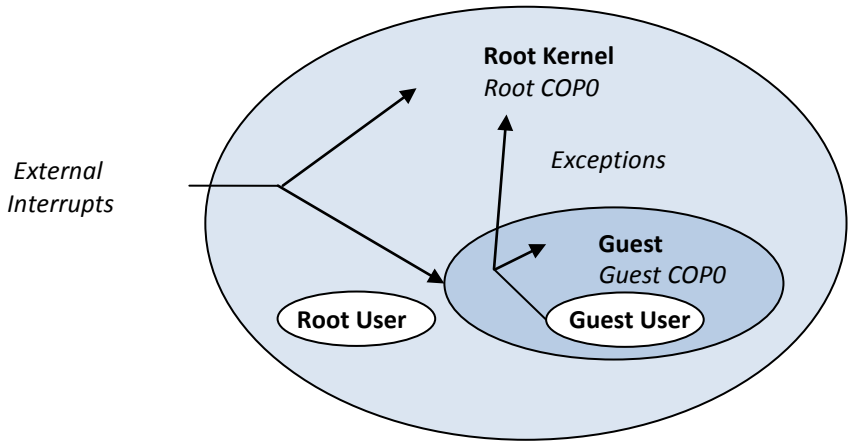

Figure 2: MIPS Virtualization ASE Onion Model

2.2.2 Replicated COP0 State for Root and Guest

In a virtualized design that supports a single copy of privileged COP0 state, the COP0 state would need to be saved to and restored from memory for context switches between root and guest. The context switches may happen due to exceptions, interrupts or root-to-guest mode changes. There are several hardware-assisted virtualization architectures that require save and restore between context switches.

To avoid the performance loss associated with context switching, the MIPS Virtualization Module duplicates the COP0 state for root and guest. Guest COP0 state is a subset of root COP0 state because certain guest actions must always be emulated by the hypervisor.

Thus, mode transitions between root and guest, which are the more frequent transitions, are not impacted by the save and restore overhead. Inter-guest context switches will continue to incur a save and restore penalty, but these switches are relatively infrequent in a typical application. It was thus deemed an appropriate tradeoff to provide only one copy of COP0 for guest operation.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

2.2.3 Support for Multiple Guests

To allow multiple guests to coexist, the MIPS Virtualization Module supports the option of a GuestID, which is a unique tag allocated by the hypervisor to each guest. The advantage of a GuestID is clear in the example of the MMU, which is shared amongst guests. In the absence of a GuestID, the hypervisor must invalidate the contents of the TLB in the MMU prior to a context switch to a different guest. Subsequently, the new guest would encounter compulsory misses. The software invalidation and compulsory misses can impact performance. With a GuestID, a guest's state can coexist in the TLB without concern for address space violations, since guests are de-aliased with the use of GuestID.

2.2.4 Independent Guest and Root MMU

In a virtualized environment, the address translation for a guest involves an additional step. A guest will first translate its virtual address (the *Guest Virtual Address* or *GVA*) to a physical address (*Guest Physical Address* or *GPA*) using the Guest TLB. This is followed by the second step where the GPA is converted into *Root Physical Address* or *RPA*, which is the system physical address.

With hardware-assisted virtualization, the benefit of having two logically separate guest and root TLBs is that the guest OS and the hypervisor can maintain separate Page Tables, and guest and root address translation can happen independently with minimal involvement of the hypervisor in the guest address translation process.

The virtualization specification allows root to access the system physical address associated with a specific guest through direct load and store access to the guest address space. Access to guest address space is convenient for example in the case where root needs to emulate guest instructions on guest exit to root on an exception.

The virtualization specification also allows for para-virtualized software to operate on the same hardware. This is done by giving full control of the guest MMU to the hypervisor by enabling trapping to root on any guest MMU operations.

2.2.5 Guest Hardware Configuration by Root

The baseline MIPS architecture provides Configuration COP0 registers that are used by an OS to determine hardware features available in the processor. In a non-virtualized system, this configuration is read-only and thus the hardware configuration is fixed.

The MIPS Virtualization Module allows root to modify guest configuration by writing the registers. In this manner, such a virtualized system can support privileged software from different hardware platforms by running them as guests with different configurations. This allows a virtualized guest to have features and capabilities which are different from the root host machine and other guests.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

2.2.6 Root Visibility into Guest Action

The MIPS Virtualization Module provides the hypervisor with visibility into guest actions. This is a requirement for any hardware-assisted virtualization solution that intends to provide full virtual privilege to a guest. Execution of *selected* privileged instructions in guest mode will cause a *Guest Privileged Sensitive Instruction (GPSI)* exception that will result in a guest exit to root. For example, if a guest is trying to modify or read the cache directly, bypassing the MMU, then such instructions will trap to root. This prevents a malicious guest from reading or modifying another guest's, or root's data in the cache.

While in guest mode, hardware events that result in setting certain COP0 fields being modified will result in a *Guest Hardware Field Change (GHFC)* exception that causes a guest exit. For example, if a guest operation detects a multiple match in the TLB, this event causes a GHFC exception. This allows the hypervisor to gracefully intervene on a guest's behalf without the guest's knowledge. Otherwise, the guest OS may panic. In such an event, the hypervisor may choose to reboot the guest without compromising other guests.

Similarly, guest software modification of certain fields will result in a *Guest Software Field Change (GSFC)* exception that causes a guest exit. For example, if a guest attempts to enable the floating-point co-processor, the hypervisor can save and restore floating-point registers in a form of lazy context-switching before allowing guest access to the floating-point co-processor in a clean state.

2.2.7 Virtualization of Interrupts

A para-virtualized solution requires that all interrupts be serviced by the hypervisor, which may adversely impact interrupt latency. Some hardware-assisted virtualization solutions similarly require that all interrupts be serviced by the hypervisor.

A MIPS Virtualization Module compliant processor allows interrupts to be selectively assigned to root or guest, where the selection is done by the hypervisor. In this case, if a guest interrupt is initially assigned to root, the MIPS architecture provides the hypervisor with the capability to inject a virtual interrupt into guest context, if it decides a guest can service the interrupt itself.

Such hardware support allows interrupts to be configured with minimal guest exits, providing native support for interrupts in guest mode of operation.

2.2.8 Native Instruction Support for Virtualization

The MIPS Virtualization Module supports instructions which allow the hypervisor to read and write the guest TLB, and read and write guest COP0 state in support of the features discussed above. These facilities allow the hypervisor to observe and modify guest TLB and COP0 state in an efficient manner.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

The MIPS' solution also provides guest software the ability to exit to root explicitly, by executing a *hypercall* instruction. This is equivalent to a system call that is executed by applications to exit to the OS in a non-virtualized system. The hypercall instruction provides a convenient means to implement an API based library for guest to call root software.

In contrast to some other architectures, the MIPS specification does not require special instructions to enter the processor into a virtualization state. If a processor is equipped with the MIPS Virtualization Module, non-virtualization software such as a single OS may operate in root mode without having to explicitly enable/disable virtualization features.

2.2.9 Debug Support

Access to debug control registers and memory is only possible through root. A guest must make an explicit call to root for debug access. The MIPS Virtualization Module supports hypervisor enabled access by a guest to debug mechanisms such as Instruction and Data Address Watch, Execution and Data Breakpoints, Trace and PC Sampling. The hypervisor can enable access for a specific guest or multiple guests at a time.

EJTAG access from an external probe station requires authentication by the hypervisor before access can be enabled.

3 MIPS Hardware-assisted Virtualization : A Complete Solution

The discussion above describes the potential advantages of hardware-assisted virtualization on a MIPS hardware platform. Though para-virtualization is appropriate for certain applications, hardware-assisted virtualization is a requirement if the application requires optimization for performance, or additional capabilities such as control over guest behavior.

The degree of hardware-assisted support may vary among other hardware-based solutions. Such solutions may not offer such a full set of features, and instead provide varying degrees of software virtualization to replace the unsupported hardware features. For example, some solutions will only virtualize address translation, but require full save and restore of the processor state on a context switch. Those that virtualize address translation may yet require hypervisor intervention for each guest TLB miss. Others may not monitor guest actions which can put a processor in an unsafe state for other guests. Or they may not virtualize interrupts, requiring hypervisor intervention for each interrupt.

4 Applications of Virtualization

Virtualization has been used on 64-bit server platforms for many years. One example of this is using virtualization to consolidate workloads from discrete platforms on to equivalent virtual machines running on a single platform. In this way, virtualization can be used to achieve significant power and space savings in the data-center, resulting in greater compute density.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

Virtualization in such an environment can also be used to support dynamic load-balancing to meet fluctuating demand, live-migration of workloads for IT maintenance, and fault-tolerance by replicating critical workloads across multiple virtual machines. For these and other applications, virtualization is thus mature and well-diversified in the enterprise space.

The MIPS Virtualization Module is also geared towards evolving requirements for virtualization-based solutions across numerous non-enterprise market segments, including mobile devices such as smartphones and tablets, and even down to small footprint, deeply embedded applications using 32-bit microcontrollers (MCUs).

The primary focus of virtualization for mobile applications is *security*. A platform is considered secure if:

1. Any application can be isolated in its own address space with absolutely no access from or to any other application's address spaces
2. A Root of Trust software base, whose integrity cannot be compromised, can be provided
3. There is a facility to provide secure services to a guest through the Root of Trust software base
4. There is the ability to isolate faults in an application by giving root visibility into a guest's actions

The Root of Trust in a virtualized platform is the hypervisor which runs in root context. The structural integrity of the hypervisor cannot be compromised as long as the system follows a *Secure Boot* process. The operational integrity is not compromised, since the hypervisor runs in its own unique context provided by the MIPS Virtualization Module, and is isolated to its own address space, which is protected by the root MMU.

In general, applications maintain address space isolation through the hypervisor-managed root MMU. The MIPS Virtualization Module also provides the capability to lock-down the contents of the root MMU at boot to provide absolute isolation of all address spaces immediately after boot. Further, in an implementation with low silicon cost requirements, the root MMU acts as a protection unit and only validates guest access without an additional address translation to a system physical address.

Root can observe a guest's actions that may put the processor in an unsafe state. This is done through hardware detection of events which in turn cause guest exit to root. The mechanisms, namely GPSI, GHFC and GSFC exceptions, were discussed in earlier sections.

Figure 3 shows a model for security on mobile platforms. A non-secure OS, practically any off-the-shelf OS, can be isolated in its own virtual machine. The non-secure OS can make an API call to the hypervisor for secure services. The secure services, such as pay-for-view media download, banking transactions, or a protected environment for corporate applications, are provided by a secure OS in a separate virtual machine.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.

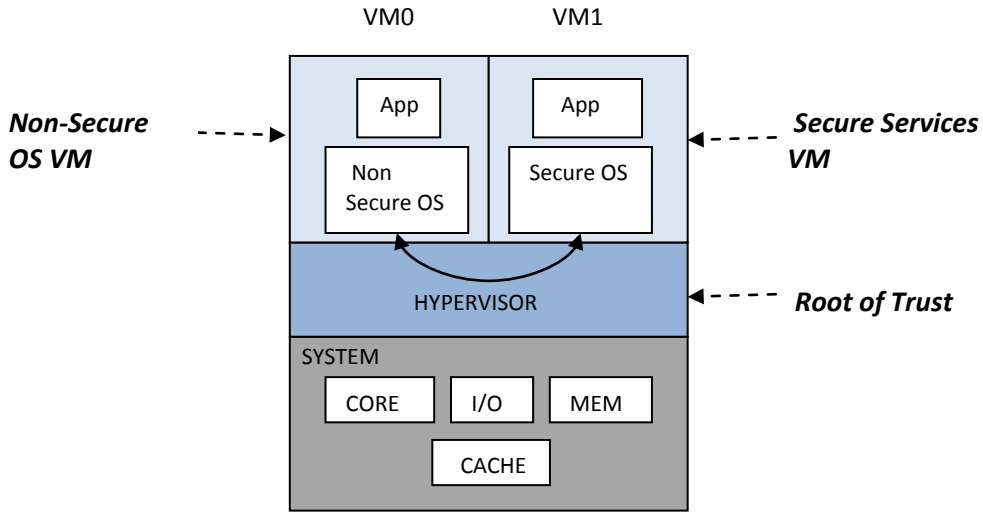

Figure 3: Secure Model for Virtualization

5 Summary

With low hardware costs, virtualization is an architectural enhancement that provides compelling benefits across a wide range of areas, from compute-intense enterprise environments to energy efficient mobile platforms. The applications are varied, from workload consolidation in the data center, to security on mobile platforms. Virtualization technology has been deployed and tested for years, and can thus be considered robust and mature. The MIPS Virtualization Module is a simple, flexible and complete hardware based solution that satisfies such varied requirements with limited or no performance impact.

Copyright © 2012 Imagination Technologies Ltd. All rights reserved.