Arm Enterprise Virtualization with Arm System IP, backplane integration and performance

By Sridhar Valluru, Arm

Over the past decade, virtualization has been used to improve system hardware utilization, reducing operating costs, and increasing security. Virtualization technology enables multiple operating systems to run in isolation on a single computer system instance, thereby reducing cost, energy consumption and physical space. Having a virtualized hardware platform also facilitates live migration of software applications from one physical system to another in the event of a hardware failure or security breach without affecting operation. Virtualization not only makes economic sense for small and medium businesses, but also adds scalability and flexibility for cloud deployment by large enterprises.

Recent trends reveal an increasing use of off-chip accelerators for offloading computationally intensive or specialized tasks to dedicated hardware. For these accelerators to operate seamlessly with on-chip processors, they must be able to address the same physical memory as the processors themselves, while retaining necessary isolation to guarantee system security. The latest Arm System Memory Management Unit (MMU-600) provides hardware-based address translation and protection capabilities for such off-chip accelerators and IO devices connected via PCIe. The MMU-600 also supports address translation caching in these devices delivering guaranteed translation performance. The Generic Interrupt Controller (GIC-600) provides interrupt translation services to route message interrupts such as MSI/MSI-X from PCIe to virtual processing elements.

Previous white papers from Arm, "Virtualization is Coming to a Platform Near You" and "Enterprise Virtualization with Arm CoreLink SMMU and Arm CoreLink GIC", discussed requirements and use cases of virtualized platforms. This white paper discusses the advances in systems IP and software support for enabling high performant virtualized machines with Arm Cortex-A processors.

Introduction

Virtualization has become ubiquitous in infrastructure applications, providing security, increasing efficiency and reducing cost by enabling multiple isolated environments concurrently on the same system hardware. Each environment runs on a virtual machine (VM) that operates under the impression of having exclusive access to the processors, peripherals, system memory and IO devices.

System performance is crucial to ensure that a virtualized environment does not affect the end user experience of the application. Performance within a virtualized environment depends on a number of factors such as: transaction bandwidth, latencies and prioritization, hypervisor and memory address translation overhead, and the number of simultaneously executing contexts. System IP from Arm takes these factors into consideration, and along with optimized drivers, delivers the best overall system performance for next generation virtualized machines.

Reference platforms and software framework from Arm are key to enabling virtualization in the Arm ecosystem. Architectural standards ensure uniformity across Arm platforms while allowing each system vendor to provide differentiation. The Linaro organization, comprising of Arm and its ecosystem partners, contributes towards building tools and drivers, which help in accelerating system development.

Virtualization support on Arm System IP

Arm System IP comprises of building blocks needed to build efficient systems around Arm v8.x cores. The IPs have evolved over multiple generations to address key performance and scalability issues to enable cloud infrastructure platforms. The key IPs enabing virtualization are the SMMU and GIC.

System Memory Management Unit (SMMU):

One of the key requirements of virtualization is to allow each VM to operate in its own address space and hardware manage access to physical memory by various system masters depending on contexts. Arm's latest System MMU, MMU-600, supports both stage-1 and stage-2 address translations with address space mapping and security mechanisms to prevent un-authorized accesses. In a typical system, stage-1 translation converts virtual address (VA) from IO masters to intermediate physical address (IPA); this address space is typically managed by the operating system. While Stage-2 translations convert from intermediate physical address (IPA) to physical address (PA); this address space is typically managed by a hypervisor. MMU-600 supports nested translations, a request for which the MMU performs both stage-1 and stage-2 translations.

 MMU-600 support's PCIe Gen4's Address Translation Service (ATS) to allow PCIe-based IO devices or accelerators (masters) to pre-fetch translations well in advance and place them in Address Translation Caches (ATC) within the IO device. With this flow, the IO master could issue DMA requests with translated addresses through the device's ATC, hence avoiding the translation overhead in the MMU. PCIe Page Request Indicator (PRI) further enhances system performance by enabling un-pinned page usage in system memory. With PRI support, IO masters pre-fetch translations that can help populate system memory pages with data from system disks without incurring adverse page fault affects This can greatly improve the efficiency of a IO master, since the disk to system memory data transfer time is no longer observed by the real transactions. Additionally, MMU enables IO masters to access a much larger range of system memory, since they can now take advantage of virtual memory.

A PCIe Root Complex (RC) with Single Root IO Virtualization (SR-IOV) function allows virtualized PCIe functions to be integrated into a system to provide IO virtualization. In a PCIe RC each virtual function (VF) and physical function (PF) pair mapping is assigned a unique PCI Express Requester ID (RID) that is mapped to a unique StreamID in the system. MMU-600 helps map virtual address to physical address using the StreamID pairs. With support for 2^{36} StreamID's, MMU-600 allows simultaneous mapping of millions of VF-PF. SR-IOV enables system traffic to bypass the software switch layer of the hypervisor virtualization stack. In a virtualized environment the VF is assigned to a virtual processing element (VPE) and the system traffic flows directly between the VF and VPE. As a result, the IO overhead in the software emulation layer is diminished, significantly reducing the overhead of a virtualized environment compared to a non-virtualized one.

MMU-600 supports hardware invalidation of translations to be propagated to ATC as defined in PCIe ATS flow to ensure IO masters always cache the most updated memory translations. Because the invalidation is handled completely by hardware, the software overhead is minimized improving overall system performance. MMU-600 is extremely scalable and can service the needs of tiny networking cards to massive server systems.

Infrastructure systems vary from a small NIC card deployed at the edge that handles streams of networking traffic from end devices to a massive server deployed at a centralized data center. Each of these system configurations typically have different performance and power/area requirements. MMU-600 design scalability allows solution providers to customize their designs to meet specific performance and area targets of the system while maintaining common MMU software drivers across the span of applications.

MMU-600 consists of two main blocks: a Translation Buffer Unit (TBU) and a Translation Cache Unit (TCU). The TBU consists of a Translation Look aside Buffer (TLB), while a central Translation Cache Unit (TCU) stores a large cache of translations and handles page table walks. A translation hit in the TBU-TLB is serviced immediately and the transaction is sent to system fabric to complete. In case of a miss in theTBU-TLB, a special transaction is sent to the TCU to fetch the address translation. The TCU cache hit returns the address tranlsation; otherwise a page table walk to system memory is performed.

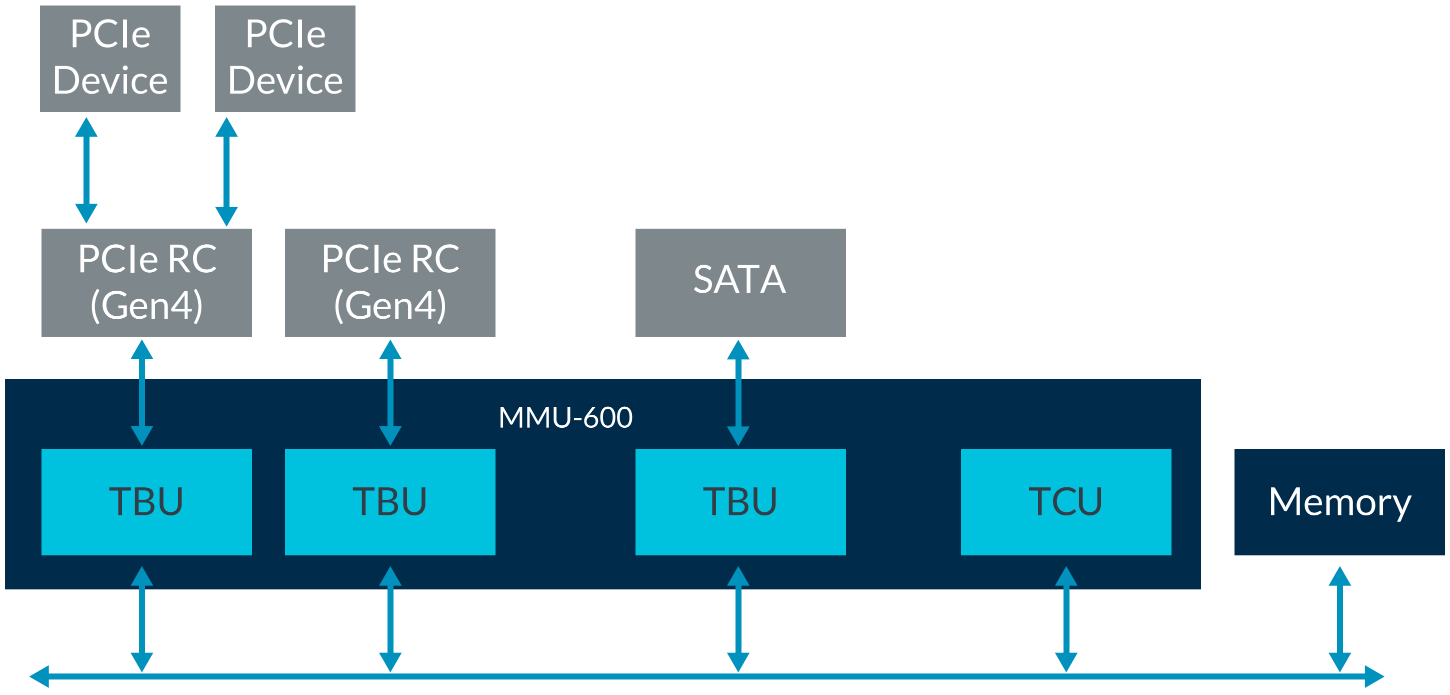

Figure 1: MMU-600 Structure

Multiple TBU's can be serviced by a single TCU as shown in Figure 1. The communication between a TBU and the TCU is over an AXI stream, using a SMMU specific protocol called Distributed Translation Interface (DTI), enabling a distributed MMU architecture. This helps design scalability by allowing TLB's to be close to IO masters and communicate with the TCU using messages over an AXI stream (or similar) backplane.

Generic Interrupt Controller (GIC):

When running operations in a virtualized environment, it is crucial to communicate interrupts and exceptions to the correct virtual processing element in a timely manner. GIC-600 is a GICv3 architectural specification compliant interrupt controller with enhanced support for a large number of cores and multiple chip configurations.

GIC-600 structurally consists of an Interrupt Translation Service (ITS) blocks, a distributor and re-distributors. The ITS block translates Message Signaled Interrupts (MSI/MSI-X) interrupts to Arm Locality-specific Peripheral Interrupts (LPI). The distributor manages interrupt routing and directs interrupts to the appropriate core-cluster that services the interrupts. The redistributor is a core specific interface and maps the incoming interrupts to specific core interrupts namely FIQ (Fast Interrupt Request), IRQ (Interrupt Request) or SError (System Error). In a virtualized environment core interrupts are virtualized and the incoming physical interrupts are mapped by a hypervisor to a VM.

GIC-600 allows interrupts to be shared and distributed across multiple compute + IO domains within a single chip. Additionally, it also supports interrupt management across multiple chips/sockets (as shown in Figure 2) or across multiple chiplets within the same package, while maintaining a unified software view of interrupts for all compute elements in the system. The inter-socket or inter-chiplet messages are ported to the native communication transport protocol supported by the system.

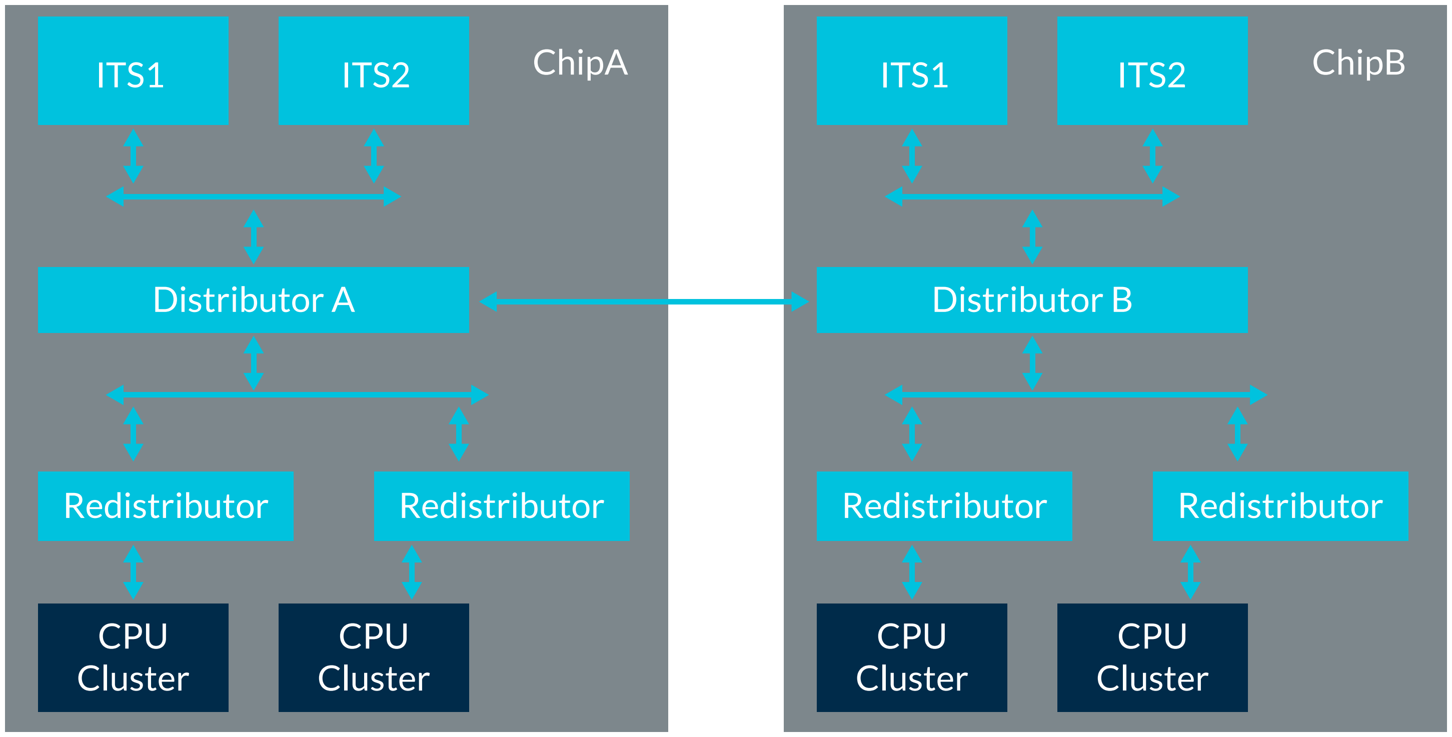

Figure 2: GIC-600 Structure

GIC-600 supports four different types of system interrupts that are mapped to the specific physical or virtual interrupts at the core:

1. Private Peripheral Interrupts (PPI): These are peripheral interrupts that target specific cores. For instance, generic timers use PPI for interrupting cores.

2. Software Generated Interrupts (SGI): These interrupts are used for inter-processor communications.

3. Shared Peripheral Interrupt (SPI): These are global peripheral interrupts that can be targetted to a specific core or to a group of cores.

4. Locality-specific Peripheral Interrupt (LPI): These are message based interrupts primarily used to translate MSI/MSI-X interrupts.

GIC-600 supports up to 56K MSI/MSI-X interrupts in a system. MSI/MSI-X translations and interrupt routing tables are stored in memory and cached by the GIC to minimize latency of recently used interrupts. Interrupts targeting a VM use the hypervisor to direct physical interrrupt to a specific VM. The GICv3 specification includes virtualization of CPU interface registers to directly control interrupts targeting a virtual processing element. The hypervisor has control of enabling/disabling the virtual CPU interface, virtual register access to enable context switching, configuring maintenance interrupts and controlling virtual interrupts. GIC-600 provides interfaces to register LPI, SPI and PPI interrupts by the hypervisor, which then identifies the target VM and priority for each interrupt. All incoming interrupts are maintained in a list register, which is used to target the interrupt to the appropriate VM. For LPIs the list is compiled using tables for each VM. With this mechanism the software overhead of mapping physical interrupts to virtual interrupts is removed.

Address Translation Performance

The primary objective of a system MMU is to provide the same view of memory to IO components in the system as the CPU itself. For this purpose, incoming transactions from these components undergo address translations from VA to IPA to PA requiring numerous lookups of the translation tables. These tables are typically stored in system memory but cached inside the MMU for performance. Whereas systems can use software to perform a VA to PA translation, this process is significantly slower compared to a hardware translation. MMU-600 enhances hardware translation performance significantly by pre-fetching and caching the translations to minimize the number of cycles spent in accessing slow system memory. This results in extremely low system performance overhead for the translations compared to a system with no MMU, and orders of magnitude improvements over translations performed by software. Figures 3 and 4 show comparisons of translation overhead on system performance for MMU-600 single and nested translations versus software translation.

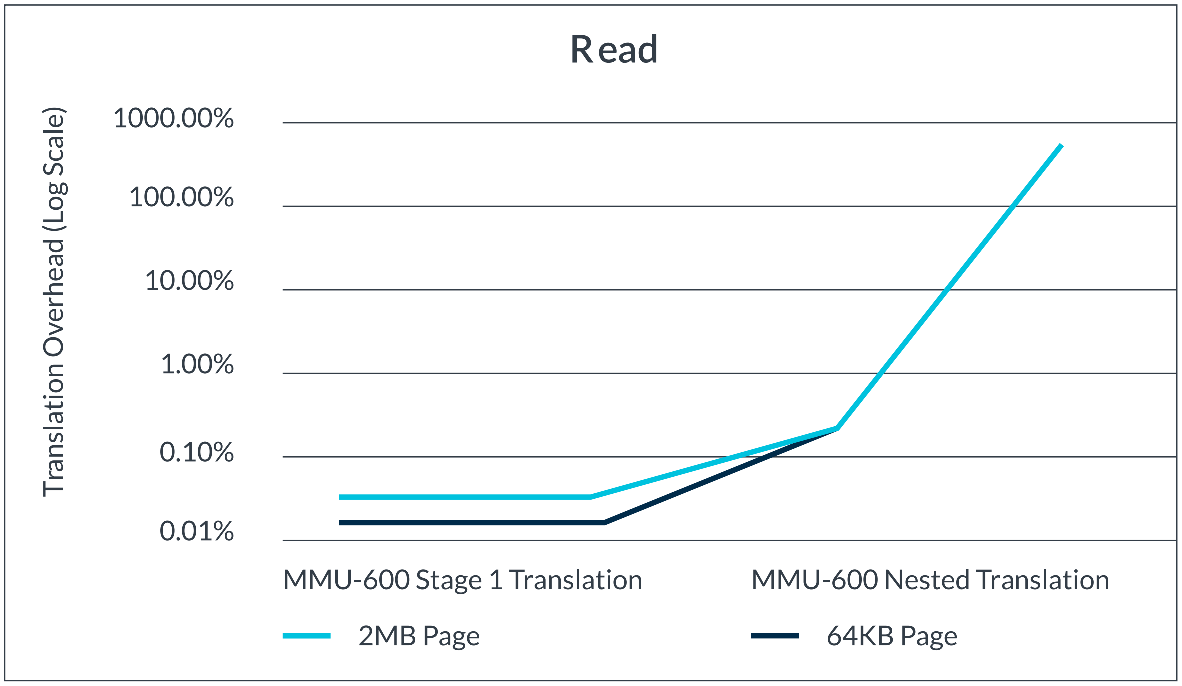

Figure 3: MMU-600 read translation overhead compared to Software (lower is better)

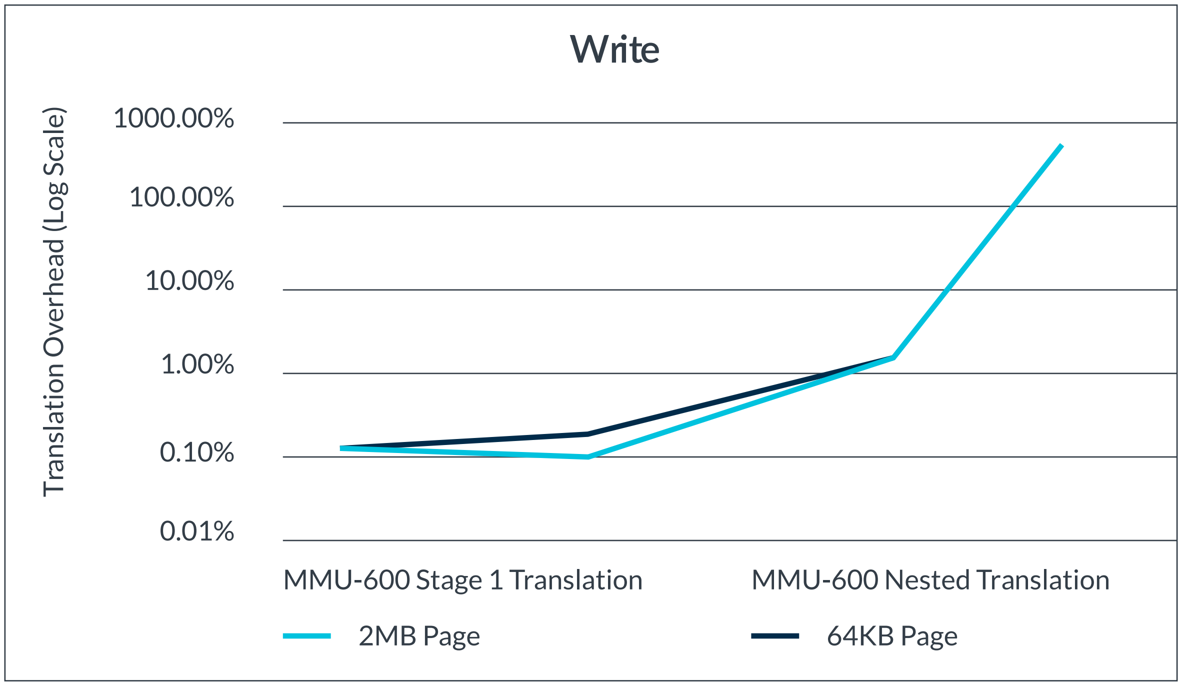

Figure 4: MMU-600 write translation overhead compared to Software (lower is better)

PCIe Flows with MMU-600 and GIC-600

Figure 5 shows a PCIe IO master interface with GIC-600 and MMU-600 System IP using a PCIe RC to translate PCIe protocol to AMBA.

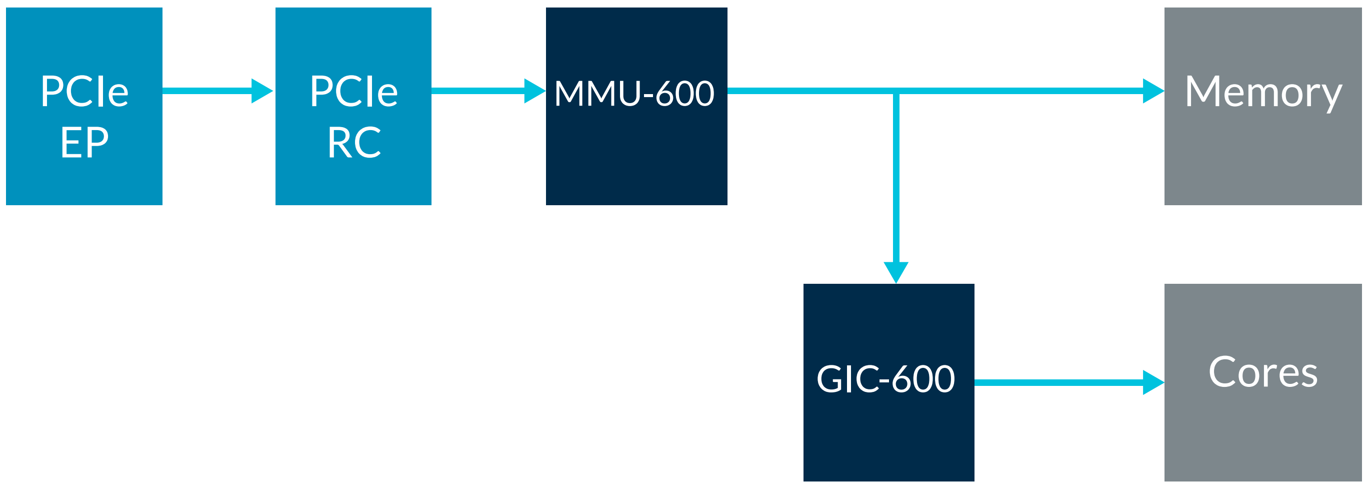

Figure 5: PCIe Integration into System

Address Translation Services (ATS) and Page Request Initiator (PRI) Flows:

To populate the Address Translation Cache (ATC) a PCIe EndPoint (EP) generates a request for address translation (ATS) to PCIe RC. PCIe RC initiates a request to MMU-600 TCU over a DTI interface to pre-fetch translations. If valid, the requested pages translations are presented to PCIe EP to populate its ATC. If the translation is not present, a negative response is reported back to the PCIe EP.

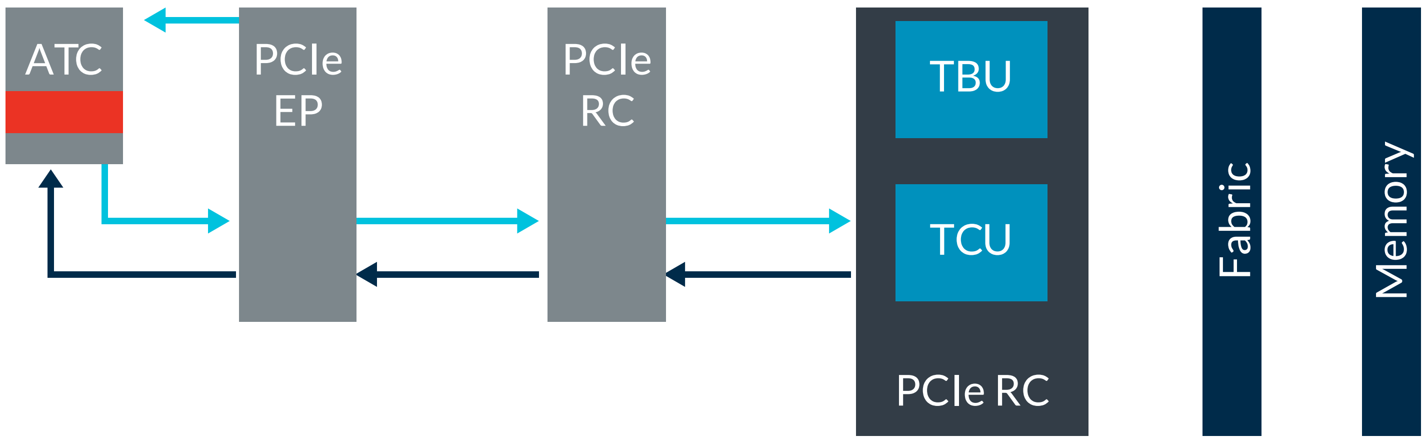

Figure 6: ATC Cache miss and ATS flow

Once the translation is available in ATC, the IO master can initiate memory access requests using the translated addresses. These requests are sent from IO master to PCIe RC. The requests are bypassed in MMU-600 TBU and sent directly to the system memory.

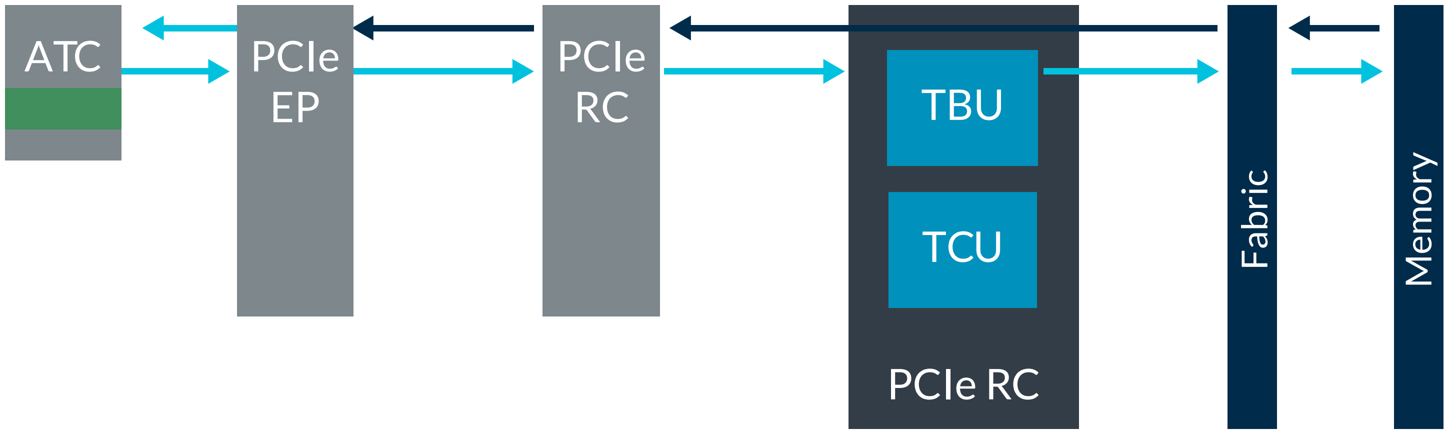

Figure 7: ATC Cache hit flow

If an ATS request fails with a not-present result, the PCIe EP could issue a Page Request Initiator (PRI) transaction to MMU-600, which asks the system software to make the requested pages resident in system memory. The system software indicates the completion of PRI process at which time the PCIe EP can initiate another ATS request to populate the ATC.

Interrupt Flow:

Interrupts from PCIe, flow in the form of Message Signaled Interrupts (MSI). The PCIe MSI's are translated to Arm LPI interrupts by the GIC-600 ITS module. The interrupts are sent to a distributor within a SoC to be routed to the appropriate physical core hosting the hypervisor. The hypervisor updates the list registers to map the incoming interrupt to a virtual interrupt targeting a specific VM..

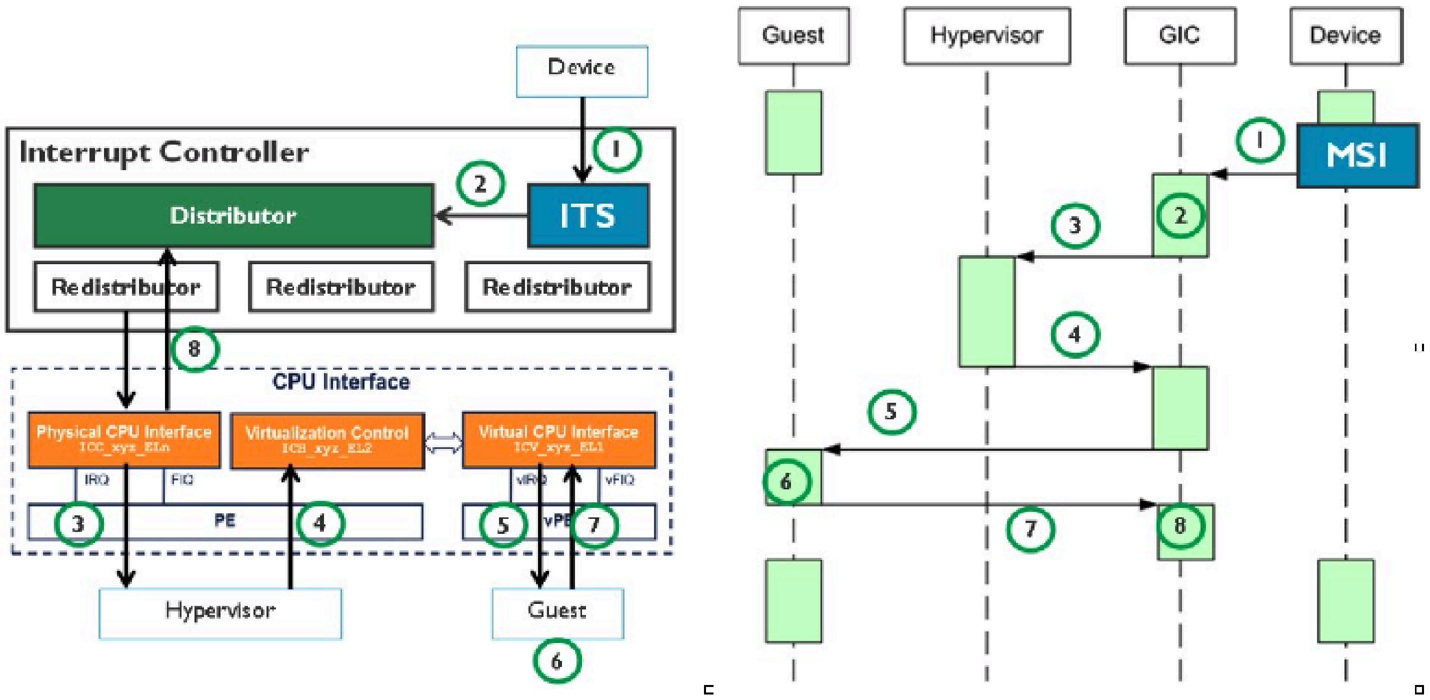

Figure 8: Interrupt flow with GIC-600

Arm Standards and Reference Subsystem

Arm provides standard specifications for ensuring system integration uniformity across various systems builders while leaving enough room for differentiation. The specifications address system architecture, boot/configuration and security/trust circles to ensure standardization of systems from various vendors and software/hardware interoperability.

Arm's Server Base System Architecture (SBSA) specification defines the key system components that are needed in every Arm-based server system to ensure a single OS image can be loaded onto any Arm partner-built system. This specification also outlines a common set of features that firmware can rely on, allowing for some commonality in firmware implementation across platforms. Fully discoverable and describable peripherals aid the implementation of such a firmware model.

Arm's Server Base Boot Requirements (SBBR) specification defines boot and runtime services that are expected by an enterprise platform OS or hypervisor. Boot and system firmware for 64-bit Arm servers is based on the Unified Extensible Firmware Interface (UEFI) specification, which defines the run-time exception level, the system memory map at boot time, the definition of a real-time clock to monitor time, and the configuration tables. Installed system hardware is defined using the Advanced Configuration and Power Interface (ACPI). In addition, various aspects of runtime system configuration, event notification and power management are also handled by ACPI. The OS must be able to use ACPI to configure the platform hardware and provide basic operations. The ACPI tables are passed, via UEFI, into the OS to drive the OSPM (Operating System-directed Power Management).

Arm's Trusted Base System Architecture (TBSA) and Trusted Base Boot Requirements (TBBR) specifications provide the framework for defining the security enclaves, secure boot components and guidance needed for robust security for server systems.

Arm reference subsystem integrates key components required for building an Arm based infrastructure platform following the standards guidelines. All software needed to boot on the reference platform is available from the Linux mainline. Key benchmarks are measured on the reference platform to aid partners in their system definition. The reference system thus allows partners to build their market specific systems on top of pre-validated platforms for faster Time-to-Market. The reference platform guide is available to Arm partners under NDA. A white paper on system performance explains Arm's approach to reference system and provides details on partner engagement.

Industry Participation

Arm works with industry partners to enable various software layers and APIs needed for virtualization. The base virtualization hypervisor technologies, KVM and Xen, have been optimized for the Arm architecture by working with open source communities.

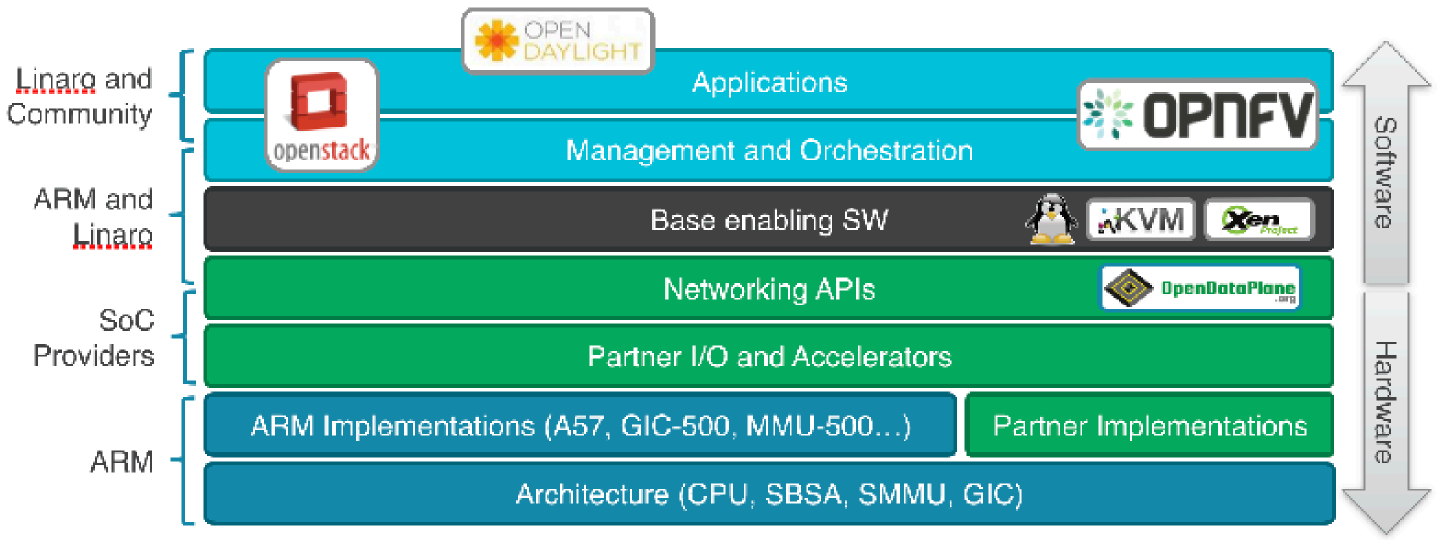

Figure 9: Arm Virtualized System Components

Linaro, an industry group within the Arm ecosystem, helps create high-level software including OpenStack, OpenNFV and OpenDataPlane. This shared software infrastructure provides an optimized foundation for Arm-based systems while reducing ecosystem fragmentation, eliminating redundant effort, and providing end users with a choice of systems from various Arm vendors.

The move towards Software Defined Networking (SDN) and Network Function Virtualization (NFV) requires high performant hardware to enable seamless transformation from customized hardware to standard hardware deployments configured by software. Arm provides key IP and works with its partners to enable this paradigm shift.

Summary

Arm provides a scalable high-performance solution for virtualization in infrastructure platforms. This solution comprises of Arm's SMMU and GIC System IP that delivers IO and interrupt management while providing the fastest path to memory. Using Arm's System IP and processors, partners can build heterogeneous platforms that seamlessly blend access to compute, IO and acceleration for virtualized workloads. Drivers for Arm System IP are supported on open-source Linux platform enabling fast deployment and guaranteed interoperability across various system implementations.

For more information on the Arm System MMU and GIC product line, please go to the system controllers' webpage: https://developer.arm.com/products/system-ip/system-controllers Additional details on Arm subsystems can be found on https://developer.arm.com/products/system-design/subsystems and https://developer.arm.com/products/system-design/system-guidance handled by ACPI. The OS must be able to use ACPI to configure the platform hardware and provide basic operations. The ACPI tables are passed, via UEFI, into the OS to drive the OSPM (Operating System-directed Power Management).

Details about Arm Server system architecture and compliance can be found at: https://developer.arm.com/products/architecture/system-architecture/server-system-architecture

More information on Linaro can be found on https://www.linaro.org/.

Trademarks

The trademarks featured in this document are registered and/or unregistered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners. For more information, visit Arm.com/about/trademarks.