Contents

[1. Stage 2 translation 1](#_Toc189671678)

[1.1. Benefits of IPA: - 2](#_Toc189671679)

[2. Emulating Memory-mapped Input/Output (MMIO) 2](#_Toc189671680)

[3. SMMU 3](#_Toc189671681)

# Stage 2 translation

Stage 2 translation allows a hypervisor to control a view of memory in a Virtual Machine (VM). Specifically, it allows the hypervisor to control which memory-mapped system resources a VM can access, and where those resources appear in the address space of the VM.

Stage 2 translations can be used to ensure that a VM can only see the resources that are allocated to it, and not the resources that are allocated to other VMs or the hypervisor.

For memory address translation, stage 2 translation is a second stage of translation. To support this, a new set of translation tables known as Stage 2 tables, are required, as shown here:

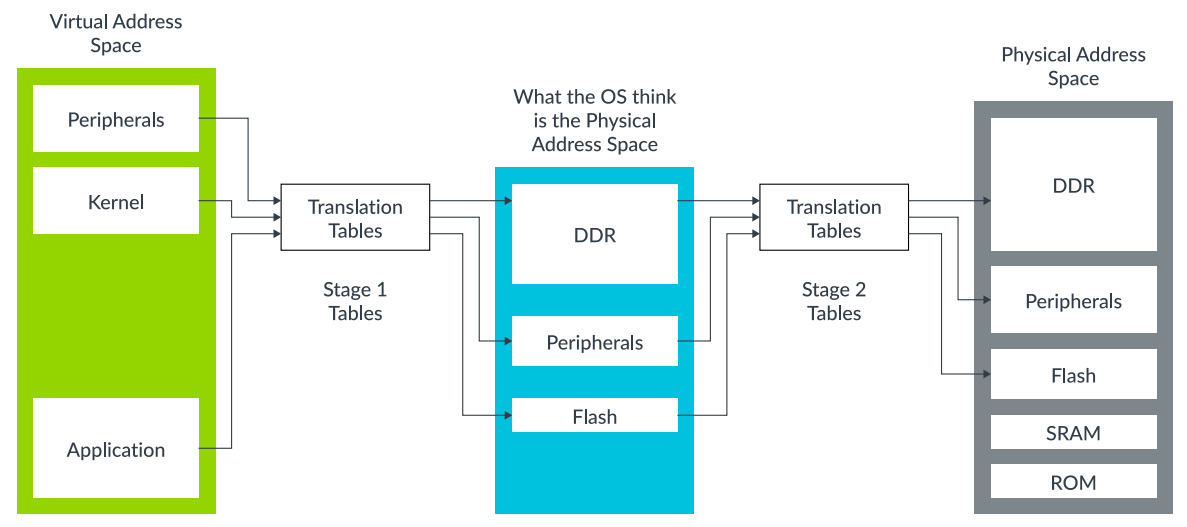


Fig - VA to IPA to PA address translation

The OS-controlled translation is called stage 1 translation, and the hypervisor-controlled translation is called stage 2 translation. The address space that the OS thinks is physical memory is referred to as the Intermediate Physical Address (IPA) space.

In ARM virtualization, "IPA" stands for "Intermediate Physical Address," which is a temporary address generated during the first stage of address translation when a virtual machine (VM) accesses memory. It allows the hypervisor to perform a second stage of translation to map the virtual address to the actual physical memory address, effectively isolating VMs from each other and managing memory access within a virtualized environment.

## Benefits of IPA: -

* Hypervisor control:

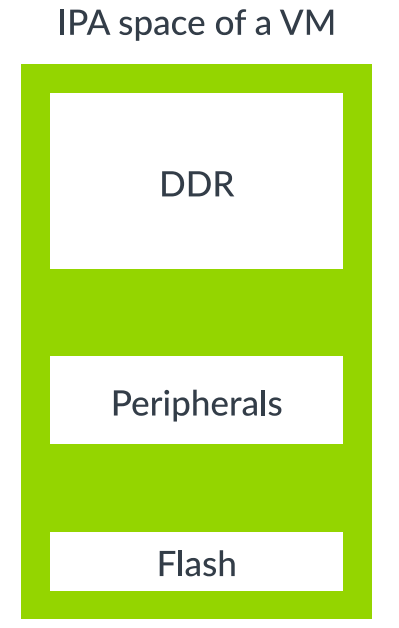
The hypervisor manages the second stage of translation using the IPA, allowing it to control memory access and security for each VM.

* Isolation:

By using separate IPA spaces for each VM, the system ensures that one VM cannot directly access the memory of another VM.

# 2. Emulating Memory-mapped Input/Output (MMIO)

Like the physical address space on a physical machine, the IPA space in a VM contains regions that are used to access both memory and peripherals, as shown here:



The VM can use peripheral regions to access both real physical peripherals, which are often referred to as directly assigned peripherals, and virtual peripherals.

Virtual peripherals are completely emulated in software by the hypervisor, as below diagram highlights:

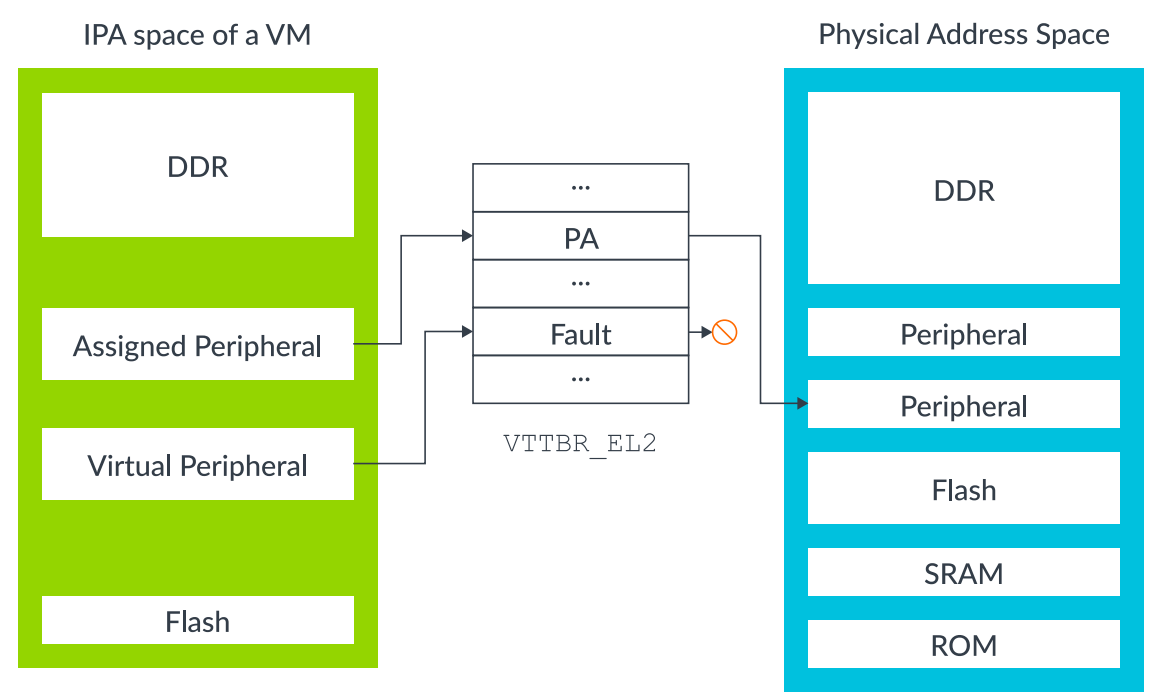


Fig - Stage 2 mappings for virtual and assigned peripherals

An assigned peripheral is a real physical device that has been allocated to the VM, and mapped into its IPA space. This allows software that is running within the VM to interact with the peripheral directly.

A virtual peripheral is one that the hypervisor is going to emulate in software. The corresponding stage 2 table entries would be marked as fault. Software in the VM thinks that it can talk directly to the peripheral, but each access triggers a stage 2 fault, with the hypervisor emulating the peripheral access in the exception handler.

# 3. SMMU

With two stages of translation, what the kernel of a Guest OS believes to be PAs are actually IPAs. The DMA controller still sees PAs; therefore the kernel and DMA controller have different views of memory. To overcome this problem, the hypervisor could trap every interaction between the VM and the DMA controller, providing the necessary translation. When memory is fragmented, this process is inefficient and problematic.

An alternative to trapping and emulating driver accesses is to extend the stage 2 regime to also cover other masters, such as our DMA controller.

Therefore, DMA controllers/masters also need an MMU. This is referred to as a System Memory Management Unit (SMMU, sometimes also called IOMMU):

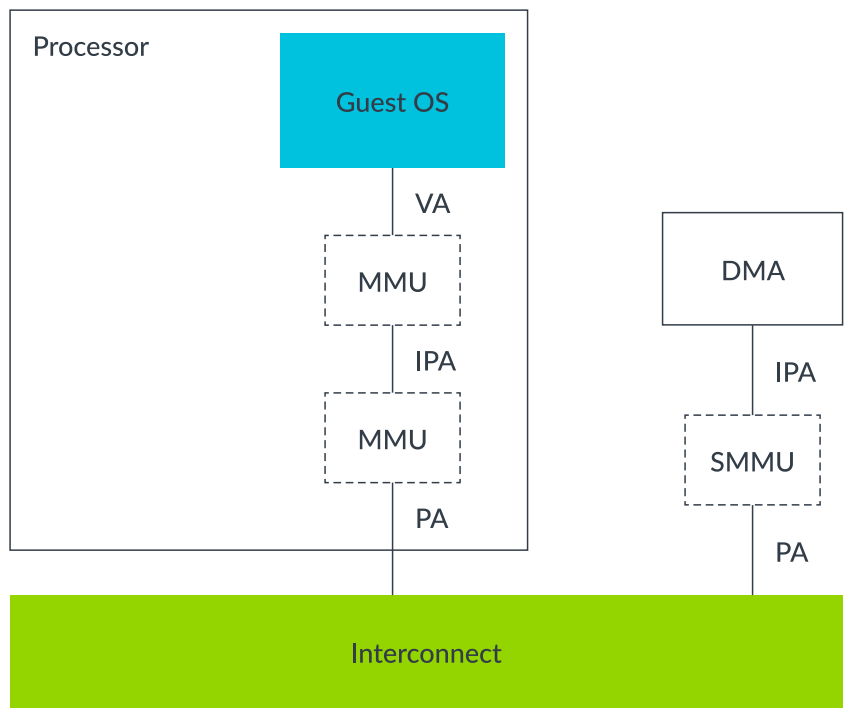


Figure - System Memory Management Unit

The hypervisor would be responsible for programming the SMMU, so that the upstream master (DMA) sees the same view of memory as the VM to which it is assigned.

The SMMU can enforce the isolation between VMs, ensuring that external masters cannot be used to breach the sandbox of VMs. The SMMU also gives a consistent view of memory to software in the VM and the external masters allocated to the VM.

Virtualization is not the only use case for SMMUs. There are many other cases that are not covered within here.

# 4. VMID

Each VM is assigned a virtual machine identifier (VMID). The VMID is used to tag translation look-aside buffer (TLB) entries, to identify which VM each entry belongs to. This tagging allows translations for multiple different VMs to be present in the TLBs at the same time.

The VMID is stored in VTTBR\_EL2 can either be 8 or 16 bits.

Note:

Translations for the EL2 and EL3 translation regimes are not tagged with a VMID, because they are not subject to stage 2 translations.

**VMID interaction with ASIDs**

TLB entries can also be tagged with an Address Space Identifier (ASID). An application is assigned an ASID by the OS, and all the TLB entries in that application are tagged with that ASID. This means that TLB entries for different applications are able to coexist in the TLB, without the possibility that one application uses the TLB entries that belong to a different application.

Each VM has its own ASID namespace. For example, two VMs might both use ASID 5, but they use them for different things. The combination of ASID and VMID is the thing that is important.

**Attribute combining and overriding**

The stage 1 and stage 2 mappings both include attributes, such as type and access permissions. The Memory Management Unit (MMU) combines the attributes from the two stages to give a final effective value.

# 5. VTTBR\_EL2

VTTBR\_EL2 (Virtualization Translation Table Base Register)

VTTBR\_EL2 is a 128-bit register that can also be accessed as a 64-bit value.

If it is accessed as a 64-bit register, accesses read and write bits [63:0] and do not modify bits [127:64].

