1. PASID (Process Address Space ID)

* allows a PCIe device to access multiple process address spaces.
* Purpose:
  + Enables memory isolation between different processes sharing a PCIe device.
  + Supports SR-IOV (Single Root I/O Virtualization) where multiple Virtual Functions (VFs) of a PCIe device serve different guest OSs or applications.
  + Used in Shared Virtual Memory (SVM), where a device can directly access CPU process memory.
* ARM Linux Usage:
  + PASID is managed by the IOMMU driver (e.g., SMMU in ARM).
  + Requires PCIe devices that support Address Translation Services (ATS) and Process Address Space ID (PASID) TLP.
  + PASID values are programmed into the IOMMU translation tables to allow a PCIe device to correctly route memory transactions.

2. SID (Stream ID)

* What it is: SID is a unique identifier assigned to a PCIe device (or function) that differentiates DMA streams.
* Purpose:
  + Allows SMMU (System MMU) to distinguish between different DMA streams.
  + Each PCIe function (physical or virtual) gets a different Stream ID. Enables per-device or per-function memory translation in the IOMMU.
* ARM Linux Usage:
  + In ARM systems, the PCIe Root Complex assigns an SID to each PCIe function.
  + The SMMU driver (e.g., arm\_smmu) uses the SID to look up translation tables.
  + Works in conjunction with PASID when multiple address spaces are needed.

Relationship Between PASID and SID

* SID is assigned at the device/function level, while PASID allows multiple address spaces per function.
* Together, they enable:
  + Fine-grained IOMMU-based memory management.
  + Efficient device sharing across multiple processes or VMs.
  + Improved security by isolating memory access.

Example in ARM Linux

When using ARM SMMU with PCIe, the kernel:

1. Assigns an SID to each PCIe function.
2. If the device supports PASID, it allows multiple address spaces per function.
3. The SMMU driver configures translation tables using these identifiers.

You can check PASID and SID in Linux with:

dmesg | grep smmu

cat /sys/kernel/debug/smmu/\*/devices

lspci -vvv | grep -i pasid

1. PASID (Process Address Space ID) in PCIe

Usage in PCIe

* PASID is a PCIe TLP (Transaction Layer Packet) extension that allows a single PCIe function to access multiple address spaces.
  + - It is used in:
      * Shared Virtual Memory (SVM): Allows a PCIe device (like a GPU or NIC) to access CPU virtual memory directly.
      * SR-IOV (Single Root I/O Virtualization): Enables Virtual Functions (VFs) to handle multiple processes with isolated memory.
      * User-space DMA (uDMA): Allows user-space applications to initiate DMA transfers without involving the kernel.

Interaction with PCIe

* Device with PASID capability:
  + The PCIe endpoint (e.g., NVMe, GPU, or network card) must support PASID.
  + The PCIe Root Complex routes PASID-tagged transactions correctly. The IOMMU (ARM SMMU) uses PASID to map transactions to the correct process address space.
* PCIe TLP Format with PASID:
  + PASID is part of the Address Translation Services (ATS) and Page Request Interface (PRI) mechanism.
  + It extends the Requestor ID (RID) to differentiate between process address spaces.

2. SID (Stream ID) in PCIe

Usage in PCIe

* SID (Stream ID) uniquely identifies a PCIe device or function at the IOMMU (SMMU) level. It enables IOMMU-based address translation for:
  + Multiple PCIe devices connected to the system. Multiple functions within a single PCIe device (SR-IOV). Efficient memory protection by allowing each stream to have its own translation settings.

Interaction with PCIe

* PCIe Device Assigns SID:
  + Each PCIe function gets a unique Stream ID. The PCIe Root Complex forwards transactions with the SID to the SMMU.
* SMMU Translates Based on SID:
  + The ARM System MMU (SMMU) uses the SID to find the correct Translation Context Bank (TCB). This determines how DMA addresses are translated into physical memory.

3. PASID + SID Together in PCIe Transactions

* SID identifies the device (or function).
* PASID identifies the process within a function.

PCIe Transaction Flow

1. A user-space application initiates a DMA transfer via a PCIe device (e.g., a GPU).
2. The PCIe endpoint (device) tags the transaction with:
   * SID (to identify the function).
   * PASID (to specify which process is requesting the DMA).
3. The PCIe Root Complex forwards the request to the ARM SMMU.
4. The SMMU checks its translation tables using SID + PASID and maps the transaction to the correct process address space.
5. The PCIe device accesses memory directly if mappings exist.

To enable PASID and SID functionality in ARM-based Linux, ensure:

* Kernel IOMMU and SMMU drivers are enabled:

CONFIG\_IOMMU\_SUPPORT=y

CONFIG\_ARM\_SMMU=y

CONFIG\_ARM\_SMMU\_V3=y

CONFIG\_PCI\_PASID=y

* The PCIe device supports PASID (lspci -vvv should list PASID capability).
* User-space drivers (e.g., for RDMA, NVMe, or GPU) support PASID-based DMA.

6. Practical Applications

|  |  |
| --- | --- |
| Application | How PASID & SID Help |
| GPU Compute | Allows multiple processes to use a single GPU with isolated memory. |
| NVMe Storage | Enables multiple virtual machines to share an NVMe SSD securely. |
| Network Offload | Allows NICs to handle multiple virtual queues independently. |
| SR-IOV | Each VF gets a unique SID, and PASID allows multiple guest OSs to share a VF. |

* In high-speed networking (e.g., RDMA over PCIe), a single NIC must serve multiple applications, VMs, or containers while ensuring memory isolation.
* Solution:
  + **SID (Stream ID): Differentiates multiple Virtual Functions (VFs) in SR-IOV-enabled NICs.**
  + **PASID (Process Address Space ID): Allows multiple user-space applications (processes) to send and receive packets without kernel involvement.**

Use Case: RDMA (Remote Direct Memory Access) over PCIe

* **PASID allows multiple applications to directly map their address space to an RDMA-capable NIC. This eliminates kernel overhead and improves latency & bandwidth.**
* **Example: In DPDK (Data Plane Development Kit), PASID is used to map multiple user-space queues to NIC hardware queues.**

RDMA + PASID Example in Linux

To check if PASID is enabled for RDMA-capable NICs:

lspci -vvv | grep -i pasid

Enable PASID in Mellanox RDMA NIC:

echo 1 > /sys/bus/pci/devices/0000:01:00.0/pasid\_enable

(Replace 0000:01:00.0 with your NIC’s PCIe address)

4. SR-IOV (Single Root I/O Virtualization) with PASID & SID

How PASID & SID Enable Secure Device Sharing

* SID (Stream ID) for SR-IOV:
  + When a PCIe device supports SR-IOV, it creates Virtual Functions (VFs) that behave like independent PCIe devices. Each VF is assigned a unique SID to ensure proper memory translation.
* PASID for Multi-Process Isolation:
  + Each VF can serve multiple processes (or VMs). PASID enables each VM (or process) to access only its assigned memory.

Example: SR-IOV with PASID on a NIC

1. Enable SR-IOV in BIOS & Kernel

echo 'options ixgbe max\_vfs=4' > /etc/modprobe.d/ixgbe.conf

modprobe -r ixgbe && modprobe ixgbe

1. Check VF Allocation

lspci -nn | grep Ethernet

1. Enable PASID for VF

echo 1 > /sys/bus/pci/devices/0000:01:00.1/pasid\_enable

SR-IOV + PASID in Virtualization (QEMU/KVM)

* In KVM, PASID allows direct passthrough of PCIe devices to VMs.
* Example QEMU command to attach a PCIe VF with PASID:

qemu-system-x86\_64 -device vfio-pci,host=0000:01:00.1,pasid=1

5. Debugging PASID & SID Issues on ARM Linux

Common Issues & Fixes

|  |  |  |
| --- | --- | --- |
| Issue | Possible Cause | Fix |
| PASID not detected in lspci | Device doesn’t support PASID | Check `lspci -vvv |
| IOMMU translation error | Incorrect PASID or SID mapping | Verify with `dmesg |
| SR-IOV VFs not appearing | BIOS or driver issue | Enable in BIOS & load correct driver (modprobe vfio-pci) |

Debugging PASID & SID Mapping

1. Check if the device supports PASID

lspci -vvv -s 0000:01:00.0 | grep PASID

1. Check IOMMU translation logs

dmesg | grep smmu

1. Verify SR-IOV VFs

ls /sys/bus/pci/devices/0000:01:00.0/virtfn\*

1. Key Components in SR-IOV with PASID in KVM

Hardware Components

* PCIe Root Complex: Manages PCIe transactions and forwards requests with PASID and Stream ID (SID).
* IOMMU (SMMU on ARM): Translates PASID-tagged memory requests to the correct guest VM's address space.
* Physical Function (PF): The main PCIe device that creates Virtual Functions (VFs).
* Virtual Function (VF): Lightweight PCIe functions created by PF to be assigned to VMs.

Software Components

* PCIe SR-IOV Driver: Creates VFs and manages PASID support in the kernel.
* VFIO (Virtual Function I/O): Provides direct device access to VMs.
* IOMMU Driver (SMMU for ARM, Intel VT-d for x86): Handles DMA translations.
* QEMU/KVM: Manages PCI passthrough and VF assignments to guest VMs.

2. PASID Implementation with Virtual Functions (VFs)

Step 1: SR-IOV Virtual Function Creation

* The PF driver in the Linux kernel enables SR-IOV and creates VFs.
* Each VF is assigned a unique Stream ID (SID) by the PCIe Root Complex.
* PASID support is enabled for each VF.

VF Creation Example

echo 4 > /sys/class/net/enp1s0/device/sriov\_numvfs

This command enables 4 Virtual Functions (VFs) on the PF.

To verify the VFs:

lspci -nn | grep Ethernet

Output:

01:00.0 Ethernet controller [8086:158b] (Physical Function)

01:00.1 Ethernet controller [8086:158b] (Virtual Function 1)

01:00.2 Ethernet controller [8086:158b] (Virtual Function 2)

Step 2: Assigning PASID to Virtual Functions

* PASID allows a single VF to handle multiple process address spaces.
* The IOMMU driver (e.g., SMMU in ARM) enables PASID support for the VFs.

Enable PASID for a VF

echo 1 > /sys/bus/pci/devices/0000:01:00.1/pasid\_enable

To check PASID capability:

lspci -vvv -s 0000:01:00.1 | grep PASID

Expected Output:

Capabilities: [100] Process Address Space ID (PASID)

Step 3: Assigning VF to a Guest VM with VFIO and QEMU

KVM uses VFIO (Virtual Function I/O) to map the VF to a VM. PASID must be enabled in VFIO and QEMU.

Bind VF to VFIO Driver

echo 0000:01:00.1 > /sys/bus/pci/devices/0000:01:00.1/driver/unbind

echo 8086 158b > /sys/bus/pci/drivers/vfio-pci/new\_id

Check if PASID is enabled in VFIO

cat /sys/kernel/iommu\_groups/\*/devices/0000:01:00.1

Launch KVM/QEMU with VF

qemu-system-x86\_64 -enable-kvm -m 4096 -cpu host -device vfio-pci,host=0000:01:00.1,pasid=1

This assigns VF 0000:01:00.1 to the guest VM with PASID enabled.

Step 4: Handling PASID in the IOMMU (SMMU/VT-d)

* When the VM initiates a DMA request, it includes a PASID in the PCIe TLP header.
* The PCIe Root Complex forwards the request with PASID & SID.
* The IOMMU (SMMU on ARM, VT-d on Intel):
  + Uses SID to identify the VF.
  + Uses PASID to locate the correct guest VM’s memory map.
  + Translates the guest virtual address to a host physical address.

Check IOMMU PASID Bindings

dmesg | grep smmu

Expected Output:

arm-smmu-v3 0000:00:00.2: SID=0x123, PASID=0x456, Guest VM memory translation enabled

3. PCIe Transaction Flow with PASID in SR-IOV

1**. Guest VM Initiates DMA**

* **The guest OS assigns a PASID to its process using iommu\_bind\_pasid\_table().**
* **The PCIe VF device receives a memory request tagged with PASID.**

**2. PCIe Root Complex Handles PASID**

* **It forwards the transaction with:** 
  + **Stream ID (SID) for the VF.**
  + **PASID for the guest process.**

**3. IOMMU Translates PASID and Maps Memory**

* **The IOMMU checks the PASID table to find the correct guest VM memory map.**
* **It remaps the guest virtual address to a host physical address.**

**4. PCIe Device Accesses Guest Memory**

* **The PCIe VF device completes the DMA transfer securely using PASID-based memory isolation.**

4. Debugging PASID Issues in KVM with SR-IOV

Check if PASID is Enabled in the Kernel

zgrep CONFIG\_PCI\_PASID /proc/config.gz

Expected Output:

CONFIG\_PCI\_PASID=y

Check IOMMU PASID Allocation

cat /sys/kernel/debug/iommu/\*/domains

Check VFIO and SR-IOV Status

dmesg | grep -i vfio

dmesg | grep -i sriov

**Based on practical experience, cache issues which I faced like invalidation in read path and flushing write path can be very usefult. Also cook up something with ATS, PRI, TPH etc.**

5. Summary of PASID + VF Implementation in KVM

|  |  |  |
| --- | --- | --- |
| Step | Component Involved | Action |
| 1 | PCIe PF Driver | Creates VFs and assigns SID |
| 2 | VFIO + SR-IOV | Binds VF to vfio-pci |
| 3 | KVM + QEMU | Assigns VF to guest VM |
| 4 | PCIe Root Complex | Tags transactions with PASID |
| 5 | IOMMU (SMMU/VT-d) | Maps PASID to guest memory |
| 6 | PCIe Device (VF) | Completes DMA with PASID isolation |

**Only if have time**

Give ARM register level details

ChatGPT said:

ARM Register-Level Details for PASID with VF in PCIe SR-IOV

In ARM-based systems, PASID (Process Address Space ID) is primarily managed through the SMMU (System Memory Management Unit) v3, which handles DMA remapping for PCIe SR-IOV devices. Below, we explore the register-level details involved in PASID processing for VFs in PCIe SR-IOV.

1. Key Components in ARM PASID Handling

1.1 Hardware Components

* PCIe Root Complex: Forwards transactions with PASID and Stream ID (SID).
* IOMMU (SMMU v3): Maps PASID-tagged memory requests to the correct guest VM.
* PCIe Physical Function (PF): Manages Virtual Functions (VFs) and assigns PASID.
* PCIe Virtual Functions (VFs): Assigned to guest VMs using VFIO + QEMU.

1.2 Key ARM Registers Involved

|  |  |  |
| --- | --- | --- |
| Register Name | Address/Offset | Description |
| SMMU\_CR0 | 0x0 | Control register to enable PASID support. |
| SMMU\_GBPA | 0x44 | Global PASID enable bit. |
| SMMU\_SIDR | 0x4 | Stream ID register for VFs. |
| SMMU\_TCRx | 0x100 + x\*8 | Translation Control Register for PASID mappings. |
| SMMU\_TTBRx | 0x200 + x\*8 | Translation Table Base Register for PASID translation. |
| SMMU\_CFGn | 0x300 + n\*8 | PASID-specific context registers. |
| SMMU\_PMCFGn | 0x400 + n\*8 | PASID table base address for VFs. |

2. Enabling PASID in SMMU (ARM IOMMU)

2.1 Enabling PASID in SMMU\_CR0

* PASID support is controlled by SMMU\_CR0 (Control Register 0).
* Bit 1 (PASIDEN) enables PASID processing.

Enable PASID in SMMU\_CR0

LDR X0, =0x0 ; Load SMMU base address

LDR X1, [X0, #0x0] ; Read SMMU\_CR0

ORR X1, X1, #(1 << 1) ; Set PASIDEN bit (bit 1)

STR X1, [X0, #0x0] ; Write back to SMMU\_CR0

2.2 Setting PASID Table Base Address in SMMU\_PMCFGn

* SMMU\_PMCFGn (Process Memory Configuration Register)
* This register holds the PASID Table Base Address.

Setting PASID Table for a VF

LDR X0, =0x400 ; Load base address of SMMU\_PMCFGn

LDR X1, =0x80000000 ; PASID Table Base Address

STR X1, [X0, #0x0] ; Write PASID Table Base

* The PASID Table is located at 0x80000000 and is used for memory translations.

2.3 Mapping PASID to VF using SMMU\_CFGn

Each PASID is mapped to a guest VM memory space via SMMU\_CFGn.

Mapping PASID 0x10 to VF 0000:01:00.1

LDR X0, =0x300 ; Load base address of SMMU\_CFGn

MOV X1, #0x10 ; PASID 0x10

STR X1, [X0, #0x0] ; Store PASID mapping

3. Handling PCIe Transactions with PASID

3.1 PCIe Transaction Format with PASID

When a guest VM accesses a PCIe VF, the PCIe Root Complex sends a request containing:

* Stream ID (SID) – Identifies the VF.
* PASID – Identifies the guest VM's process.
* I/O Virtual Address (IOVA) – Address to be translated.

3.2 Translation Flow

|  |  |  |
| --- | --- | --- |
| Step | Component | Action |
| 1 | Guest VM | Issues DMA request with PASID |
| 2 | PCIe Root Complex | Tags transaction with PASID & SID |
| 3 | SMMU (IOMMU) | Looks up PASID table in SMMU\_PMCFGn |
| 4 | SMMU | Translates guest virtual address to host physical address |
| 5 | PCIe Device (VF) | Accesses translated memory |

4. Debugging PASID Issues at Register Level

4.1 Checking PASID Enable Status

Read SMMU\_CR0 to verify PASID support:

LDR X0, =0x0

LDR X1, [X0, #0x0] ; Read SMMU\_CR0

TST X1, #(1 << 1) ; Check PASIDEN bit

B.EQ PASID\_DISABLED

4.2 Checking PASID Table Base

Read SMMU\_PMCFGn to check PASID Table:

LDR X0, =0x400

LDR X1, [X0, #0x0] ; Read PASID Table Base Address

4.3 Checking PASID-SID Mapping

Read SMMU\_CFGn to verify PASID mapping:

LDR X0, =0x300

LDR X1, [X0, #0x0] ; Read PASID configuration

5. Summary of ARM PASID Implementation

|  |  |  |
| --- | --- | --- |
| Component | Register | Function |
| Enable PASID | SMMU\_CR0 (bit 1) | Enables PASID processing |
| Set PASID Table | SMMU\_PMCFGn | Defines PASID-to-memory mapping |
| Assign PASID to VF | SMMU\_CFGn | Maps PASID to Stream ID (SID) |
| Check PASID Transactions | SMMU\_TTBRx | Translates PASID-addressed memory |

**PASID Translation in SMMU (ARM IOMMU) - Page Table Format and Implementation**

In ARM-based systems, PASID (Process Address Space ID) enables per-process I/O virtualization in PCIe SR-IOV devices. The System Memory Management Unit (SMMU) v3 handles PASID-tagged DMA transactions by mapping them to guest VM memory spaces. Below is a deep dive into the page table format and how PASID translation works at the register level.

1. PASID Address Translation Flow

When a PCIe Virtual Function (VF) issues a DMA request with a PASID, the SMMU translates it to a physical address using the following steps:

1. PCIe Device Issues PASID-Tagged DMA Request
   * PCIe VF sends a memory request tagged with PASID.
   * Root Complex forwards it to SMMU (IOMMU).
2. SMMU Identifies PASID Context Table
   * Uses SMMU\_PMCFGn (Process Memory Configuration Register) to fetch the PASID table base.
   * PASID is used as an index in the PASID table.
3. PASID Table Entry Points to PASID Context Descriptor
   * The PASID Context Descriptor contains:
     + SMMU\_TTBR0 (L1 Page Table Base)
     + SMMU\_TCR0 (Translation Control Register)
     + SMMU\_MAIR0 (Memory Attributes)
     + Address Space Identifier (ASID)
4. SMMU Walks the Page Table Hierarchy
   * Based on the TTBR0/TTBR1, SMMU performs Page Table Walk.
   * Translates Guest IOVA → Host Physical Address (HPA).
5. Transaction is Forwarded to Memory Controller
   * The translated address is sent to the memory controller.

2. **PASID Table Format in SMMU**

The PASID Table is a structure indexed by PASID values, pointing to PASID Context Descriptors.

PASID Table Structure

|  |  |  |
| --- | --- | --- |
| Offset | Field | Description |
| 0x00 | PASID Context Descriptor Pointer | Points to the PASID-specific address space |
| 0x08 | SMMU\_TTBR0 | L1 Page Table Base Address |
| 0x10 | SMMU\_TCR0 | Translation Control (granule size, ASID) |
| 0x18 | SMMU\_MAIR0 | Memory attributes |

Example: PASID Table Entry

Each PASID maps to a context descriptor:

|  |  |
| --- | --- |
| PASID | Context Descriptor Address |
| 0x10 | 0x8000\_0000 |
| 0x11 | 0x8000\_1000 |
| 0x12 | 0x8000\_2000 |

Example Memory Layout in SMMU\_PMCFGn:

Base Address: 0x8000\_0000

PASID 0x10 → Context Descriptor at 0x8000\_0000

PASID 0x11 → Context Descriptor at 0x8000\_1000

PASID 0x12 → Context Descriptor at 0x8000\_2000

3. PASID Context Descriptor (Memory Space Mapping)

Each PASID Context Descriptor contains:

* SMMU\_TTBR0/TTBR1 → Points to the Guest VM's page tables.
* SMMU\_TCR0 → Defines address translation settings.
* SMMU\_MAIR0 → Defines memory attributes.

PASID Context Descriptor Format

|  |  |  |
| --- | --- | --- |
| Offset | Register | Description |
| 0x00 | SMMU\_TTBR0 | L1 Page Table Base |
| 0x08 | SMMU\_TTBR1 | L2 Page Table Base |
| 0x10 | SMMU\_TCR0 | Translation settings |
| 0x18 | SMMU\_MAIR0 | Memory Attributes |
| 0x20 | SMMU\_ASID | Address Space ID |
| 0x28 | SMMU\_FSR | Fault Status |

Example: Setting PASID Context Descriptor for a Guest VM

1. Set PASID Table Base (SMMU\_PMCFGn)

LDR X0, =0x400 ; Load SMMU\_PMCFGn base

LDR X1, =0x80000000 ; PASID Table Base Address

STR X1, [X0, #0x0] ; Store PASID Table Base

2. Configure PASID Context Descriptor

LDR X0, =0x80000000 ; PASID 0x10 Context Descriptor

LDR X1, =0x90000000 ; Guest VM's L1 Page Table Base (TTBR0)

STR X1, [X0, #0x00] ; Store TTBR0

LDR X1, =0x91000000 ; Guest VM's L2 Page Table Base (TTBR1)

STR X1, [X0, #0x08] ; Store TTBR1

LDR X1, =0x4000 ; Memory attribute settings

STR X1, [X0, #0x18] ; Store SMMU\_MAIR0

LDR X1, =0x20 ; ASID (Address Space ID)

STR X1, [X0, #0x20] ; Store SMMU\_ASID

4. Page Table Walk for PASID Translation

Step 1: L1 Page Table Walk

* SMMU reads TTBR0 (0x90000000)
* Finds L1 Page Table Entry → Points to L2 Page Table.

Step 2: L2 Page Table Walk

* Reads TTBR1 (0x91000000)
* Finds Physical Page Address → Returns Host Physical Address.

Step 3: Return Translated Address

* Returns Host Physical Address (HPA) to PCIe device.

5. Debugging PASID Page Table Translation

Check PASID Table Base

LDR X0, =0x400

LDR X1, [X0, #0x0] ; Read PASID Table Base

Check PASID Context Descriptor

LDR X0, =0x80000000

LDR X1, [X0, #0x00] ; Read TTBR0

LDR X1, [X0, #0x08] ; Read TTBR1

LDR X1, [X0, #0x10] ; Read TCR0

Check Page Table Entries

LDR X0, =0x90000000 ; Read L1 Page Table

LDR X1, [X0, #0x0] ; Fetch L2 Page Table Entry

6. Summary of PASID Address Translation

|  |  |  |
| --- | --- | --- |
| Step | Component | Action |
| 1 | PASID Table (SMMU\_PMCFGn) | Maps PASID to Context Descriptor |
| 2 | PASID Context Descriptor | Stores TTBR0, TTBR1 for page table lookup |
| 3 | Page Table Walk | L1 → L2 → Final Physical Address |
| 4 | Translated Address | Returned to PCIe device |

1. PCIe TLP Structure with PASID

A Transaction Layer Packet (TLP) consists of several fields:

General PCIe TLP Structure

|  |  |  |
| --- | --- | --- |
| Field | Bits | Description |
| Type | 5 | Defines request type (e.g., Memory Read, Memory Write) |
| TC (Traffic Class) | 3 | Priority classification |
| Attributes | 2 | No Snoop, Relaxed Ordering |
| Length | 10 | Number of DWORDs |
| Requester ID | 16 | Bus/Device/Function number |
| Tag | 8 | Identifies unique transactions |
| Address | 64 | Target memory address |
| PASID (Optional) | 20 | Identifies process address space |

PASID Encoding in TLP Headers

PCIe Extended Capability (PASID Capability) adds PASID in Address Translation Services (ATS) or Shared Virtual Memory (SVM) transactions.

|  |  |  |
| --- | --- | --- |
| TLP Field | Standard TLP | TLP with PASID |
| Type | Mem Read | Mem Read + PASID |
| Length | 64-bit | 64-bit |
| Address | 0xXXXX\_XXXX\_XXXX | 0xXXXX\_XXXX\_XXXX |
| PASID | *Not present* | 0x12345 |

2**. PCIe PASID TLP Flow in SR-IOV**

1. **VF Issues Memory Read with PASID**
   * **PCIe VF issues a TLP Memory Read with PASID in the header.**
   * **Example: A GPU VF accesses guest memory tagged with PASID.**
2. **Root Complex (RC) Forwards TLP to SMMU**
   * **RC decodes PASID and forwards it to SMMU/IOMMU for address translation.**
3. **SMMU Uses PASID Table for Translation**
   * **PASID Table maps PASID → Guest VM Page Table.**
   * **Performs VA → PA translation and updates TLP with physical address.**
4. **Memory Controller Processes Request**
   * **The memory controller services the translated memory address.**
5. **Completion TLP Sent Back to VF**
   * **Memory completion response is sent back to the VF, maintaining PASID.**

3. Example: PCIe Memory Read with PASID

VF to Host Memory Read Request

TLP Type: Memory Read with PASID

PASID: 0x12345

Virtual Address: 0x4000\_0000

Request Length: 64 bytes

Root Complex forwards to SMMU:

PASID: 0x12345 → Lookup PASID Table

Translation: 0x4000\_0000 (Guest) → 0xA000\_0000 (Host)

SMMU updates TLP with translated address:

TLP Type: Memory Read

PASID: 0x12345 (Maintained)

Physical Address: 0xA000\_0000

Completion Response:

TLP Type: Memory Read Completion

PASID: 0x12345

Data: <64 bytes>

4. Debugging PASID Transactions in PCIe

Checking PASID Capability in PCIe Devices

Use lspci to verify PASID support:

lspci -vvv -s 0000:03:00.0 | grep -i pasid

Expected Output:

Capabilities: [A0] PASID: Enabled

Tracing PASID Transactions in Kernel Logs

Enable PASID debug logs:

echo 1 > /sys/kernel/debug/iommu/verbose

dmesg | grep PASID

Checking PASID Registers in a PCIe Device

Dump PCIe configuration space:

setpci -s 03:00.0 0x180.L

Output:

0x00012345 (PASID Enabled)

5. Summary

|  |  |
| --- | --- |
| Step | Action |
| 1 | VF sends TLP with PASID |
| 2 | Root Complex forwards to SMMU |
| 3 | SMMU translates PASID to Physical Address |
| 4 | Translated request sent to memory controller |
| 5 | Completion TLP returned to VF |

**Only if time**

PASID Translation in SMMU (ARM IOMMU) – Page Table Format and Implementation

In ARM-based systems, PASID (Process Address Space ID) enables per-process I/O virtualization in PCIe SR-IOV devices. The System Memory Management Unit (SMMU) v3 plays a crucial role in translating PASID-tagged DMA requests by mapping them to guest VM memory spaces.

1. PASID Address Translation Flow in SMMU

When a PCIe Virtual Function (VF) issues a DMA request with a PASID, the SMMU translates it to a physical address using these steps:

1. VF Sends PASID-Tagged DMA Request
   * PCIe VF issues a Memory Read/Write request with a PASID.
   * This TLP is forwarded to Root Complex (RC).
2. Root Complex Sends TLP to SMMU
   * RC detects PASID-enabled translation and forwards the request to SMMU.
3. SMMU Looks Up PASID Context Table
   * The PASID Table Base Address is stored in SMMU\_PMCFGn.
   * The PASID is used as an index in this table to fetch the PASID Context Descriptor.
4. SMMU Reads PASID Context Descriptor
   * Contains key registers:
     + SMMU\_TTBR0 (L1 Page Table Base)
     + SMMU\_TCR0 (Translation Control)
     + SMMU\_MAIR0 (Memory Attributes)
     + SMMU\_ASID (Address Space ID)
5. SMMU Walks Page Table to Translate Address
   * The Guest Virtual Address (GVA) is translated into a Host Physical Address (HPA).
6. Request is Forwarded to the Memory Controller
   * The final translated physical address is used for the DMA operation.

2. PASID Table Format in SMMU

The PASID Table is indexed by PASID values, and each entry contains a PASID Context Descriptor.

PASID Table Layout

|  |  |  |
| --- | --- | --- |
| Offset | Field | Description |
| 0x00 | PASID Context Descriptor Pointer | Points to the PASID-specific memory space |
| 0x08 | SMMU\_TTBR0 | L1 Page Table Base Address |
| 0x10 | SMMU\_TCR0 | Translation Control (granule size, ASID) |
| 0x18 | SMMU\_MAIR0 | Memory attributes |

Example: PASID Table Entry

Each PASID maps to a context descriptor:

|  |  |
| --- | --- |
| PASID | Context Descriptor Address |
| 0x10 | 0x8000\_0000 |
| 0x11 | 0x8000\_1000 |
| 0x12 | 0x8000\_2000 |

PASID Table Memory Layout:

Base Address: 0x8000\_0000

PASID 0x10 → Context Descriptor at 0x8000\_0000

PASID 0x11 → Context Descriptor at 0x8000\_1000

PASID 0x12 → Context Descriptor at 0x8000\_2000

3. PASID Context Descriptor (Memory Space Mapping)

Each PASID Context Descriptor contains:

* SMMU\_TTBR0/TTBR1 → Guest VM page tables.
* SMMU\_TCR0 → Translation settings.
* SMMU\_MAIR0 → Memory attributes.

PASID Context Descriptor Format

|  |  |  |
| --- | --- | --- |
| Offset | Register | Description |
| 0x00 | SMMU\_TTBR0 | L1 Page Table Base |
| 0x08 | SMMU\_TTBR1 | L2 Page Table Base |
| 0x10 | SMMU\_TCR0 | Translation settings |
| 0x18 | SMMU\_MAIR0 | Memory Attributes |
| 0x20 | SMMU\_ASID | Address Space ID |
| 0x28 | SMMU\_FSR | Fault Status |

Example: Setting PASID Context Descriptor for a Guest VM

1. Set PASID Table Base (SMMU\_PMCFGn)

LDR X0, =0x400 ; Load SMMU\_PMCFGn base

LDR X1, =0x80000000 ; PASID Table Base Address

STR X1, [X0, #0x0] ; Store PASID Table Base

2. Configure PASID Context Descriptor

LDR X0, =0x80000000 ; PASID 0x10 Context Descriptor

LDR X1, =0x90000000 ; Guest VM's L1 Page Table Base (TTBR0)

STR X1, [X0, #0x00] ; Store TTBR0

LDR X1, =0x91000000 ; Guest VM's L2 Page Table Base (TTBR1)

STR X1, [X0, #0x08] ; Store TTBR1

LDR X1, =0x4000 ; Memory attribute settings

STR X1, [X0, #0x18] ; Store SMMU\_MAIR0

LDR X1, =0x20 ; ASID (Address Space ID)

STR X1, [X0, #0x20] ; Store SMMU\_ASID

4. Page Table Walk for PASID Translation

Step 1: L1 Page Table Walk

* SMMU reads TTBR0 (0x90000000)
* Finds L1 Page Table Entry → Points to L2 Page Table.

Step 2: L2 Page Table Walk

* Reads TTBR1 (0x91000000)
* Finds Physical Page Address → Returns Host Physical Address.

Step 3: Return Translated Address

* Returns Host Physical Address (HPA) to PCIe device.

5. Debugging PASID Page Table Translation

Check PASID Table Base

LDR X0, =0x400

LDR X1, [X0, #0x0] ; Read PASID Table Base

Check PASID Context Descriptor

LDR X0, =0x80000000

LDR X1, [X0, #0x00] ; Read TTBR0

LDR X1, [X0, #0x08] ; Read TTBR1

LDR X1, [X0, #0x10] ; Read TCR0

Check Page Table Entries

LDR X0, =0x90000000 ; Read L1 Page Table

LDR X1, [X0, #0x0] ; Fetch L2 Page Table Entry

6. Summary of PASID Address Translation

|  |  |  |
| --- | --- | --- |
| Step | Component | Action |
| 1 | PASID Table (SMMU\_PMCFGn) | Maps PASID to Context Descriptor |
| 2 | PASID Context Descriptor | Stores TTBR0, TTBR1 for page table lookup |
| 3 | Page Table Walk | L1 → L2 → Final Physical Address |
| 4 | Translated Address | Returned to PCIe device |