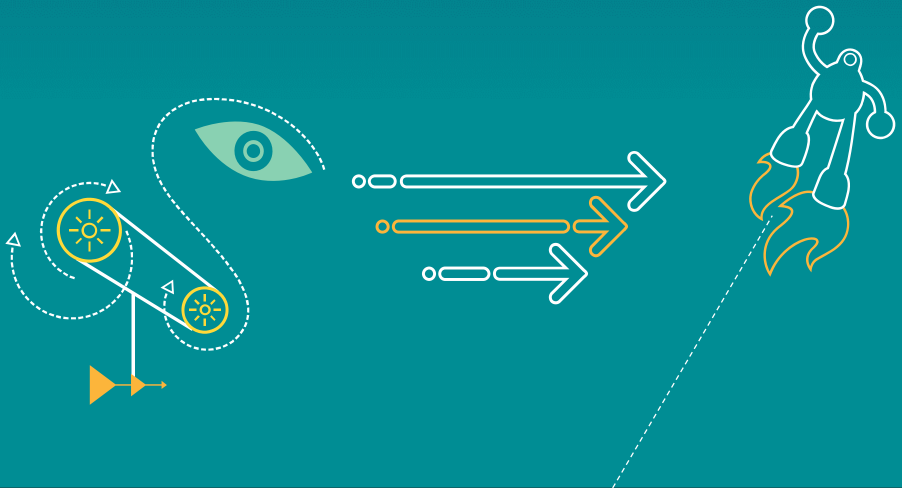
System Memory Management Unit (SMMU) Overview

Qualcomm Technologies, Inc.



80-P2055-1 B

Revision History

Revision	Date	Description
A	November 2015	Initial release
B	December 2015	Slide 19 was added

Contents

SMMU Overview

- SMUU is basically an MMU for devices
- SMMU hardware block allows virtually contiguous memory to be backed by physically noncontiguous pages
- Memory translation logic in the SMMU is the same as the logic in the CPU MMU
- SMMU and IOMMU are used interchangeably

Key Terminology

- Context Bank
  - Set of registers that defines a translation context; every context bank in the system has the same set of registers
- Stream ID
  - Identifier used as input to the SMMU to uniquely identify the current transaction stream
- IOVA
  - I/O Virtual Address, a.k.a DMA address and Virtual address
- Page Tables
  - Tables in memory that define the virtual-to-physical mappings; SMMU walks these tables to perform translations
- TTBR0
  - Register holding the base address of the page tables

Key Terminology (cont.)

- Page Level
  - Level of the page table walk; generally only comes up when looking at translation faults
- Translation Stage
  - Stage of the translation process; Stage 1 is managed by HLOS; Stage 2 is managed by the hypervisor
- IPA
  - Intermediate Physical address; output address from Stage 1 and input address to Stage 2
- TLB
  - Translation Lookaside Buffer; cached IOVA to Phys mapping
- Bypass
  - When a stage (stage 1, stage 2, or both) is not considered for translation (i.e., input = output)

Important APIs

API	Description
arm_iommu_create_mapping	configure a VA range
arm_iommu_attach_device	associate a VA range with a device and enable SMMU translations
dma_alloc_coherent	allocate buffer and map into SMMU for device
dma_map_sg	map existing buffer (by **struct scatterlist**) into SMMU for device
ion_share_dma_buf	Convert an Ion handle to a dma buf
dma_buf_attach	Prepare a dma buf for dma on the given device
dma_buf_map_attachment	Get the scatterlist for the given attachment
msm_iommu_get_bus	Get proper bus for your domain

Device Tree Setup

- Top level device tree nodes for SMMUs. Clients will use phandles to this node in their config

File: arch/arm64/boot/dts/qcom/msm-arm-smmu.dtsi

```
jpeg_smmu: arm,smmu-jpeg@fda64000 {
        compatible = "qcom,smmu-v2";
        /* ... */
};
```

- Client node (for a device driver that only needs one context bank)

Example:

```
qcom,my_device@fda1c000 {
        compatible = "qcom,my-driver";
        /* ...your usual driver code here... */
        iommus = <&jpeg_smmu 0>,
                         <&jpeg_smmu 1>;
};
```

- With this DT setup, device pointer for qcom, my_device will be used everywhere that a device pointer is required for DMA APIs.

Device Tree Setup (cont.)

- Client node (for a device driver that needs multiple context banks)

```
qcom,my_device@fda1c000 {
        compatible = "qcom,my-driver";
        /* ...your usual driver code here... */

        /* context bank "sub-devices": */
        iommu_test_device_cb1 {
                compatible = "qcom,my-driver-new-cb";
                iommus = <&jpeg_smmu 0>,
                         <&jpeg_smmu 1>;
        };

        iommu_test_device_cb2 {
                compatible = "qcom,my-driver-new-cb";
                iommus = <&jpeg_smmu 2>;
        };
};
```

- With this setup, the device pointers associated with the "sub-device" DT nodes will be used for the various DMA APIs.

API Usage: Mapping a buffer into an SMMU with DMA APIs

- Define the virtual address range to be used for allocations

  Example:

  ```
  struct dma_iommu_mapping *mapping;
  dma_addr_t va_start = 0;
  size_t va_len = 0x40000000;
  sturct bus_type *bus;

  bus = msm_iommu_get_bus(dev);  // Pass the context bank device pointer here
  as discussed on the previous slides
  mapping = arm_iommu_create_mapping(bus, va_start, va_len);
  ```

  This allocates memory for the page table

- After creating the structure dma_iommu_mapping, associate the mapping with the device:

```
arm_iommu_attach_device(cb_device, mapping);  // context bank device pointer
                                                 and mapping to be passed
```

API Usage: Allocating and Mapping a New Buffer

- Allocate a buffer which will be mapped into the SMMU:

  Example:

```
size_t size = 0x1000;
void *cpu_addr;
dma_addr_t iova;
cpu_addr = dma_alloc_coherent(cb_device, size, &iova, GFP_KERNEL);
```

  where, cb_device is your context bank device pointer

- dma_alloc_coherent() automatically maps the buffer into the SMMU if arm_iommu_attach_device() is called for this device before calling dma_alloc_coherent().

API Usage: Mapping an Existing Buffer

- If there is an existing buffer that was allocated in some other way besides dma_alloc_* (e.g. Ion memory, kmalloc), one can map the buffer into the SMMU with the dma_map_* APIs
  - struct dma_buf* is needed

```
struct dma_buf *buf = ion_share_dma_buf(client, handle);      // if we have a
struct ion_handle
OR
struct dma_buf *buf = dma_buf_get(fd);  // through a dma_buf file descriptor
```

  - Once there is a struct dma_buf* available, map it into the SMMU as follows:

```
struct dma_buf_attachment *attach =    dma_buf_attach(buf, cb_device);
struct sg_table *table =  dma_buf_map_attachment(attach, DMA_TO_DEVICE);

dma_map_sg(cb_device, table->sgl, table->nents, DMA_TO_DEVICE);
```

where cb_device is the context bank device pointer.

- Supported values for the direction argument to dma_map_sg() are mapped to IOMMU page protection attributes as follows:

DMA_TO_DEVICE	Device can read from the memory
DMA_FROM_DEVICE	Device can write to the memory
DMA_BIDIRECTIONAL	Device can read from or write to the memory

API Usage: Mapping an Existing Buffer (cont.)

- After mapping, the output address (which is given to the device) can be obtained with sg_dma_address(table->sgl) and the length can be obtained with sg_dma_len(table->sgl).
- After one is finished with the buffer, it must be unmapped

```
dma_unmap_sg(cb_device, table->sgl, table->nents, DMA_TO_DEVICE);
dma_buf_unmap_attachment(attach, table, DMA_TO_DEVICE);
dma_buf_detach(dma_buf, attach);
dma_buf_put(dma_buf);
```

- **Clocks and Power**
  - SMMU clients need to control clocks and power on MSM8996
    - SMMU Clients *do not* need to enable clocks and power when they call into the SMMU driver (through the DMA APIs) to set up mappings.
    - SMMU Clients *do* need to enable clocks and power when they expect the SMMU to be servicing transactions.

SMMU Faults

- Two types of SMMU Faults
  - Global Fault
  - Context Fault
- Global Fault
  - Occurs when either:
    - SMMU translation, when being processed, has no associated translation context bank

    or
    - A translation context bank is not the appropriate place to record the fault.
  - Fatal by default
  - Shows up in TZ/Hypervisor logs
  - Examples:
    - Unidentified stream faults: no match for the given stream ID was found in the stream matching table
    - Configuration Access Faults: Missing clocks, power.

SMMU Faults (cont.)

- Context Fault
  - Associated with a particular translation context bank
  - Categorized as Stage-1 and Stage-2 context faults
  - Stage-1 context faults are non-fatal and show up in the kernel log
  - Stage-2 context faults are fatal and show up in the hypervisor log
  - Examples
    - Translation Fault: An unmapped address was accessed
    - Permission Fault: Attempt to write to read-only memory, fetch instructions from non-executable memory, etc.

Important SMMU registers

- FSR
  - Fault Status Register; identifies the type of fault (translation, permission etc.)
- FAR
  - Fault Address Register; provides the input address that caused the fault
- FSYNRm
  - Fault Syndrome Registers; provides more information about the faulting access
- CBFRSYNRA
  - Context Bank Fault Restricted Syndrome Register A; provides the SID
- TTBRm
  - TTBR0 holds the base address of translation table 0
  - TTBR1 holds the base address of translation table 1

Fault Status Register

- Context Faults
  - Bit 0: Reserved
  - Bit 1: Translation Fault
  - Bit 2: Access Fault
  - Bit 3: Permission Fault
  - Bit 4: External fault
  - Bit 5: Match conflict fault
  - Bit 6: TLB Lock Fault
  - Bit 8: Address Size Fault
  - Bit 9: Unsupported upstream transaction

- Global Faults
  - Bit 0: Invalid context fault
  - Bit 1: Unidentified Stream ID Fault
  - Bit 2: Stream Match Conflict Fault
  - Bit 3: Unimplemented Context Bank Fault
  - Bit 4: Unimplemented Context Interrupt
  - Bit 5: Configuration Access Fault
  - Bit 6: External Fault
  - Bit 7: Permission Fault
  - Bit 8: Unsupported Upstream Transaction

Other Registers (Important Bits)

- Context Bank Fault Restricted Syndrome Register (CBFRSYNRAn)
  - Gives fault syndrome information about the access that caused an exception in the associated translation context bank.
  - StreamID- Bits[15:0]
    - StreamID of the transaction that caused the fault
- Context Bank Fault Syndrome Register (CBn_FSYNRn)
  - Holds fault syndrome information about the memory access that caused a synchronous abort exception
  - Bit[4] = WNR -- Write Not Read. The possible values of this bit are:
    - 0 Read.
    - 1 Write

SMMU Faults Examples

A) Context Fault from Venus (TZ log)

```
SMMU:>> VENUS CB8 Fault:
FSR=0x40000402                    >> Translation Fault
FAR=0x00000000004e0000      >> Faulting address = 0x4e0000
IPAFAR=0x00000000004e0000
FSYNR0=0x00080023
FSYNR1=0x170c0000
CBFRSYNRA8=0x00010180         >> SID = 0x180 (lower 16 bits)
…..
```

B) Context Fault from Display (TZ log)

```
SMMU:>> MDP CB2 Fault:
 FSR=0x40000408                   >> Permission Fault
 FAR=0x00000000048a1000     >> Faulting address = 0x48a1000
 IPAFAR=0x0000000000000000
 FSYNR0=0x00000027
 FSYNR1=0x1c03000a
 CBFRSYNRA2=0x00000000
 ……
```

C) Global Fault from Venus (TZ log)

```
SMMU GLOBAL CLIENT NON-SEC FAULT: bit mask=0x00000002
SMMU:>> VENUS NonSec Global Fault:
 NSGFSR=0x80000002              >> FSR : Unidentified Stream ID Fault
 NSGFAR=0x00000000deadd000
 NSGFSYNR0=0x0000000c
 NSGFSYNR1=0x00000006
 NSGFSYNR2=0x17000000
 NSCR0=0x00201e36
```

References

Documents	
Resources	
ARM System Memory Management Unit Architecture Specification	http://infocenter.arm.com/help/topic/com.arm.doc.ihi0062d.b/IHI0062D_b_system_mmu_architecture_specification.pdf

Questions?

https://support.cdmatech.com