Rockchip Developer Guide Linux IOMMU

ID: RK-KF-YF-110

Release Version: V1.0.0

Release Date: 2019-12-23

Security Level: □Top-Secret □Secret □Internal ■Public

Fuzhou Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

IOMMU is used for the conversion of 32-bit virtual addresses and physical addresses. It has read-write control bits and can generate page fault exceptions and bus exception interrupts.

Product Version

Chipset	Kernel Version
All chipset	4.4 & 4.19

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers
Software development engineers

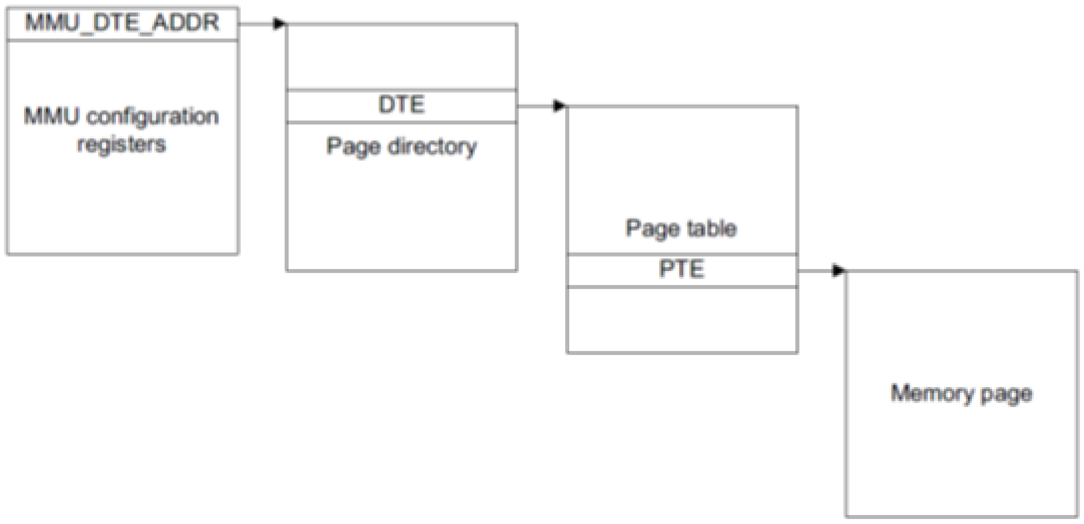
Revision History

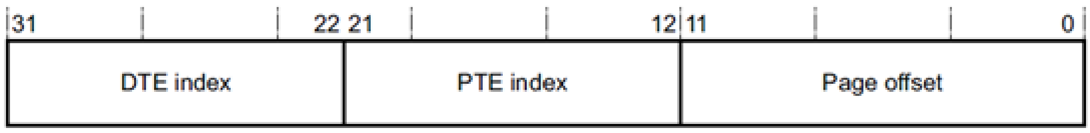
Version	Author	Date	Change Description
V1.0.0	Simon.Xue	2019-12-23	Initial version

Rockchip Developer Guide Linux IOMMU

1. IOMMU Structure

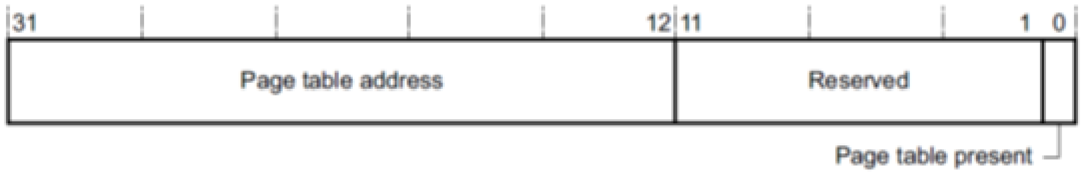
It is using 2 level table structure, which as follow:





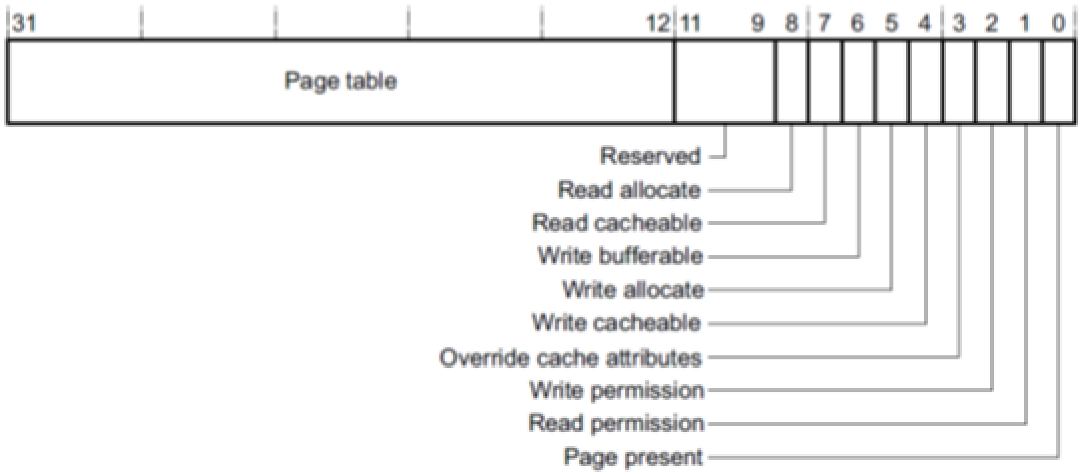
32bit address structure, the first 10 bits of the first-level page table offset, the middle 10 bits of the second-level page table offset, and the last 12 bits within the page offset.

DTE structure：



bit0：indicates whether the next page table present

PTE structure：



bit0：indicates whether the actual physical page present

bit1：Read permission

bit2：write permission

2. IOMMU Driver

2.1 Driver File

The driver file is in:
`drivers/iommu/rockchip-iommu.c`

2.2 DTS Configuration

The reference DTS configuration is `Documentation/devicetree/bindings/iommu/rockchip,iommu.txt` here introduce the follow parameter mainly:

- `compatible = "rockchip,iommu";`

For all the iommu of the device, the compatible field value is the same

- `interrupts = <GIC_SPI 119 IRQ_TYPE_LEVEL_HIGH 0>;`

It used for exceptional interrupt, such as page fault interrupts.

- `clocks = <&cru ACLK_VOP1>, <&cru HCLK_VOP1>;`
- `clock-names = "aclk", "hclk";`

IOMMU and master share clock, here IOMMU driver controls clock separately

- `power-domains = <&power RK3399_PD_VOPL>;`

IOMMU driver manipulates PD.

- `iommu-cells = <0>;`

  Here value must be 0, the reason refer to iommu.txt

3. IOMMU Usage

The ROCKCHIP IOMMU driver depends on the IOMMU framework (`drivers/iommu/iommu.c`), which mainly implements the callback function in `struct iommu_ops rk_iommu_ops`, and then the master calls the API provided by the iommu framework to operate on iommu, as follows:

1. iommu attach

   `iommu_attach_device -> rk_iommu_attach_device` /* enable iommu */

2. iommu detach

   `iommu_detach_device -> rk_iommu_detach_device` /* disable iommu */

3. iommu map

   `iommu_map -> rk_iommu_map`

   Create a page table and establish the mapping relationship between the virtual address and the physical address. When debugging, open the dbg in iommu_map and observe the mapping

4. iommu unmap

   `iommu_unmap -> rk_iommu_unmap`

   Remove the mapping relationship between the virtual address and the physical address, and release the virtual address space. When debugging, open the dbg in iommu_unmap and observe
   unmapping

5. domain alloc

   `iommu_domain_alloc -> rk_iommu_domain_alloc`

   Apply page table base address for attach / detach operation

6. domain free

   `iommu_domain_free -> rk_iommu_domain_free`

   Free page space

7. dump iommu

   Take RK3399 `vopl_iommu` as example, assume the current virtual address is 0x00001000, dump page table as follow order

   1. obtain the level 1 page table base address: DT

      io -4 0xff8f3f00

   2. calculate page level 1 page table offset

      index1 = VA >> 22

   3. calculate page level 1 page table physical address: DTE

      DTE = index1 * 4 + DT

   4. obtain the level 2 page table base address: PT

      PT = io -4 DTE

   5. calculate page level 2 page table offset

      index2 = VA && 0x3ff000

   6. calculate page level 1 page table physical address: PTE

      PTE = index2 * 4 + PT

   7. obtain PAGE physical address: page

      page = io -4 PTE

   8. Calculate in-page offset

      offset = page + (VA && 0xfff)

   offset is the physical address corresponding to the virtual address 0x00001000, which the master can use to analyze whether the data is correct

8. dma-mapping

   1. if dev is not iommu device

      ARM32: `dev->dma_ops = arm_dma_ops;`

      ARM64: `dev->dma_ops = arm64_swiotlb_dma_ops;`

2. if dev is iommu device

   ARM32: `dev->dma_ops = iommu_ops`;

   ARM64: `dev->dma_ops = iommu_dma_ops`;

take `dma_alloc_attrs` as example:

1. For non-iommu dev, call alloc callback from a's dma_ops to alloc continuous physical memory and kernel mode virtual address
2. For iommu dev, call the alloc callback from b's dma_ops to alloc physical memory, and call it through the iommu framework `iommu_map` to create the IOMMU page table, establish the mapping between the virtual address and the physical address, and return the first IOMMU virtual address and kernel mode virtual address

One of the easiest steps to use IOMMU

```
1. domain = iommu_domain_alloc(&platform_bus_type);
2. iommu_map(domain, iova, paddr, size, prot);
3. iommu_attach_device(domain, dev);
4. master access memory via iommu
```

IOMMU is a basic component that can be embedded in various memory allocation frameworks, such as ION / DRM. Taking DRM under the ARM64 environment as an example, a complete IOMMU buffer allocation and mapping process is as follows

```
rockchip_gem_alloc_buf ->
rockchip_gem_get_pages ->
rockchip_gem_iommu_map ->
iommu_map_sg ->
iommu_map
```

The IOMMU mapping process by passing FD is as follows:

```
struct dma_buf *dmabuf = dma_buf_get(fd) ->
dma_buf_attach -> dma_buf_map_attachment ->
map_dma_buf -> drm_gem_map_dma_buf ->
dma_map_sg_attrs -> map_sg ->
__iommu_map_sg_attrs ->
iommu_dma_map_sg ->
iommu_map_sg ->
iommu_map
```

4. Kernel configuration

```
Symbol: ROCKCHIP_IOMMU [=y]
Type  : boolean
Prompt: Rockchip IOMMU Support
    Location:
        -> Device Drivers
            -> IOMMU Hardware Support (IOMMU_SUPPORT [=y])
    Defined at drivers/iommu/Kconfig:211
    Depends on: IOMMU_SUPPORT [=y] && (ARM || ARM64 [=y]) && (ARCH_ROCKCHIP [=y]
||
                COMPILE_TEST [=n])
    Selects: IOMMU_API [=y] && ARM_DMA_USE_IOMMU
```

5. IOMMU FAQ

1. Pagefault interrupt

   A pagefault interrupt occurs, indicating that the current IOMMU has a page fault exception, that is, the virtual address currently being accessed does not create a matched map. There are caused by three possibilities, the first one is to access the address not mapped, the other is access beyond the mapping area, and the third is to start accessing without mapping. In history, three situations above all have appeared in master.

2. Error IOMMU enable stall

   This is likely to be that a pagefault exception has occurred in IOMMU, the master does not handle the exception but continue to visit, which can be find from the log.

3. Error access IOMMU registers

   It is likely caused by the master's processing of PD, that is, the use of `pm_runtime_get_sync / pm_runtime_put_sync` is unreasonable, which also means accessing to the IOMMU register without opening the IOMMU power domain.

4. Continue to trigger IOMMU interrupt

   The IOMMU interrupt number is incorrect in DTS file.

5. Splash screen

   During vop display, enable IOMMU may leads to vop access memory error. In the chip without frame effect function, should not enable IOMMU until vop is in idle status.

6. Error IOMMU register

   It is very likely caused by the master accesses the IOMMU register out of IOMMU register range or the master resets the entire IP.

7. Device Link

   IOMMU integrates device link operation and hands over the PD operation authority to the master. The master needs to pay attention to the use of `pm_runtime_get / pm_runtime_put`.

8. Shared IOMMU

   In the ARM32 environment, the master of the shared IOMMU needs to maintain
   independent page tables, such as VEPU and VDPU. Before each accessing, the matched page table needs to be attached. ARM64 is a shared page table, and does not need to be attached every time.