**DMA缓冲区**

* **说明**
  + DMA缓冲是从内存中专用于DMA的内存区域分配得来的。
* **分配DMA缓冲区的方法**
  + 在系统启动时，分配DMA缓冲区
  + 在运行时，分配DMA缓冲区
  + 为缓冲区保留顶部物理内存
* **注意**
  + 并不是所有的 **内存区间** 都适合DMA操作
  + 一些设备和一些系统中的**高端内存**不能使用DMA，这是因为外围设备不能使用**高端内存**的地址
* 使用DMA的设备
  + 连接到总线接口的硬件(DMA控制器，DMA控制器和设备相连)
  + 总线和内存相连
  + 设备驱动程序操作的是内核虚拟地址
  + 设备驱动程序和硬件进行通信
  + 用户空间程序要访问DMA缓冲区的数据，需要建立内核虚拟内存和用户空间虚拟内存的映射
* 桥接
  + **接口总线**的I/O地址 和 **桥接电路**的物理地址建立映射关系
* **通用DMA层**
  + 并不是所有的系统都能对全部的内存执行DMA
  + 因此，出现了通用的DMA层，在这一次实现了很多操作DMA的接口，隐藏了细节，呈现出与总线，体系架构无关的通用的接口。
* **DMA层的函数接口**
  + 判断设备在指定的体系架构下的寻址能力的函数接口
    - int dma\_set\_mask(struct device \*dev, u64 dma\_mask);
    - mask: 32位(0xFFFFFFFF)；24位(0xFFFFFF)；16位(0xFFFF)
    - 返回值：非0，该设备能够在指定的体系架构下使用DMA；0:表示不能
    - 默认情况下，内核执行DMA是基于32位的。
* **内核逻辑地址和总线地址相互转换的接口**
  + 内核逻辑地址 转换成 总线地址
    - unsigned long virt\_to\_bus(void \*x);
  + 总线地址 转换成 逻辑地址
    - void \*bus\_to\_virt(unsigned long x);
* **DMA映射的组成**
  + DMA缓冲区（用户空间地址）
  + DMA缓冲区地址（内核逻辑地址）
  + 设备可访问的地址（即是总线地址，设备直接和总线通信，然后总线在内存上进行数据的存取）
* **IOMMU（I/O内存管理单元）**
  + 规划了物理内存，使得物理上分散的缓冲区，对于设备看来是连续的
  + 并不是所有的体系架构都支持IOMMU
* **回弹缓冲区**
  + 驱动程序执行的DMA的内存地址，对于外围设备来说是不可访问的时候，如高端内存，这个时候，就需要创建回弹缓冲区了
  + 也就说，设备和内存 的数据访问，多了一个中间层，回弹缓冲区，对于设备来说数据的读写，都是对于回弹缓冲区来进行的。
* **DMA层表示总线地址的类型**
  + dma\_addr\_t（typedef u32 dma\_addr\_t;）
  + 该类型对驱动是不透明的，也就是说驱动不能直接操作它
  + 如设备本身是可以操作的程序的生命周期中。
* **两种类型的DMA映射**
  + 一致性DMA映射
    - 又名 共享DMA映射，存在于驱动
    - 可以同时被CPU和外围设备访问
    - 建立一致性映射的函数接口
      * void \***dma\_alloc\_coherent**(struct device \*dev, size\_t size, dma\_addr\_t \*dma\_handle,int flag)
        + 获得的映射最小为单个页
        + 作用是：

分配DMA缓冲区

建立DMA缓冲区的映射

* + - * + 参数

size:DMA缓冲区的大小

dma\_handle:保存返回的总线地址

flags:GFP\_分配方式

* + - * + 返回值

返回缓冲区的内核虚拟地址，这个值是

* + - 回收建立的一致DMA缓冲区
      * void dma\_free\_coherent(struct device \*dev, size\_t size, void \*cpu\_addr, dma\_addr\_t dma\_handle)
  + 流式DMA映射
    - 单独的DMA操作，不会和其他共享的，创建流式DMA映射
    - 建立流式映射，必须告诉内核数据流动的方向

enum dma\_data\_direction {

DMA\_BIDIRECTIONAL = 0, //数据双向移动

DMA\_TO\_DEVICE = 1, //数据发送到设备

DMA\_FROM\_DEVICE = 2, //数据发送到CPU

DMA\_NONE = 3, //只用于调试目的

};

* + - 一个缓冲区要传输数据的时候，使用的接口
* **DMA池**
  + DMA池 是一个生成小型，**一致性DMA映射**的机制。
    - DMA池可以有小于单个页大小的缓冲池
    - 创建DMA池的函数接口
      * struct dma\_pool \*dma\_pool\_create (const char \*name, struct device \*dev, size\_t size, size\_t align, size\_t allocation)
        + 参数

name : DMA池的名字

size: 创建的缓冲池的大小

align: 硬件对齐规则上

allocation：非0表示内存边界不能超越allocation，也就是从该DMA池中分配缓冲区时，分配的缓冲区不能超过allocation这个上限值

* + - 释放DMA池
      * void dma\_pool\_destroy (struct dma\_pool \*pool)
    - 从DMA缓冲池中分配DMA缓冲区
      * void \*dma\_pool\_alloc (struct dma\_pool \*pool, int mem\_flags, dma\_addr\_t \*handle);
        + 参数

mem\_flags:GFP\_分配方式

handle：保存的总线地址

* + - * + 返回值

分配的DMA缓冲区的内核虚拟地址

* + - 释放从DMA缓冲池中分配的缓冲区
      * void dma\_pool\_free (struct dma\_pool \*pool, void \*vaddr, dma\_addr\_t dma);