CONFIG_HAVE_GENERIC_DMA_COHERENT => drivers/base/dma-coherent.c

CONFIG_DMA_CMA => drivers/base/dma-contiguous.c
CONFIG_DMA_CMA depends on CONFIG_CMA (=> mm/cma)

```
struct device {
......

    struct dma_coherent_mem *dma_mem; /* internal for coherent mem
                            override */
#ifdef CONFIG_DMA_CMA
    struct cma *cma_area;       /* contiguous memory area for dma
                           allocations */
#endif

......
};
```

dma_mem is for dma-coherent.c
cma_are is for dma-contiguous.c

这两个fields用于分配per-device dma memory

1. drivers/base/dma-mapping.c
2. arch/arm/mm/dma-mapping.c
3. drivers/base/dma-coherent.c
4. drivers/base/dma-contiguous.c

(1) arch-independent dma-mapping
(2) arch-dependent dma-mapping (ARM)
(3)(4) arch-independent per-device dma memory management

dma-coherent.c会处理dtb中reserved memory.

> RESERVEDMEM_OF_DECLARE(dma, "shared-dma-pool", rmem_dma_setup);

for example

```
        linux,cma {
            compatible = "shared-dma-pool";
            reusable;
            size = <0x0 0x10000>;
            alignment = <0x0 0x2000>;
            linux,cma-default;
        };
```

rmem_dma_setup()的作用是设定

> rmem->ops = &rmem_dma_ops;

```
1.  static const struct reserved_mem_ops rmem_dma_ops = {
2.      .device_init    = rmem_dma_device_init,
3.      .device_release = rmem_dma_device_release,
4.  };
```

而 `.device_init` and `.device_release` 则是在drivers/of/of_reserved_mem.c被调用。

```
1.  /**
2.   * of_reserved_mem_device_init() - assign reserved memory region to given de
    vice
3.   *
4.   * This function assign memory region pointed by "memory-region" device tree
5.   * property to the given device.
6.   */
7.  int of_reserved_mem_device_init(struct device *dev)
8.  {
9.      struct reserved_mem *rmem;
10.     struct device_node *np;
11.     int ret;
12.
13.     np = of_parse_phandle(dev->of_node, "memory-region", 0);     ①
14.     if (!np)
15.         return -ENODEV;
16.
17.     rmem = __find_rmem(np);                                      ②
18.     of_node_put(np);
19.
20.     if (!rmem || !rmem->ops || !rmem->ops->device_init)
21.         return -EINVAL;
22.
23.     ret = rmem->ops->device_init(rmem, dev);                     ③
24.     if (ret == 0)
25.         dev_info(dev, "assigned reserved memory node %s\n", rmem->name);
26.
27.     return ret;
28. }
```

of_reserved_mem_device_init()用于处理在dtb中为specific device指定的专属dma memory. 即当该device需要alloc dma memory时，会从这块reserved dma memory中分配。

in arch/arm/boot/dts/vexpress-v2m.dtsi

```
clcd@1f000 {
    compatible = "arm,pl111", "arm,primecell";
    reg = <0x1f000 0x1000>;
    interrupt-names = "combined";
    interrupts = <14>;
    clocks = <&v2m_oscclk1>, <&smbclk>;
    clock-names = "clcdclk", "apb_pclk";
    memory-region = <&v2m_video_ram>;
    max-memory-bandwidth = <50350000>; /* 16bpp @ 25.175MHz */

    port {
        v2m_clcd_pads: endpoint {
            remote-endpoint = <&v2m_clcd_panel>;
            arm,pl11x,tft-r0g0b0-pads = <0 8 16>;
        };
    };

    panel {
        compatible = "panel-dpi";

        port {
            v2m_clcd_panel: endpoint {
                remote-endpoint = <&v2m_clcd_pads>;
            };
        };

        panel-timing {
            clock-frequency = <25175000>;
            hactive = <640>;
            hback-porch = <40>;
            hfront-porch = <24>;
            hsync-len = <96>;
            vactive = <480>;
            vback-porch = <32>;
            vfront-porch = <11>;
            vsync-len = <2>;
        };
    };
};
};
```

```
        v2m_video_ram: vram@3,00000000 {
            compatible = "arm,vexpress-vram";
            reg = <3 0x00000000 0x00800000>;
        };
```

clcd应该是LCD controller,该device使用的专属与它的memory.该块memory也应该是被reserved。

①

> memory-region = <&v2m_video_ram>;

获得 `v2m_video_ram` node

②

`v2m_video_ram` node的memory应该是被reserved.其已经被记录在

> static struct reserved_mem reserved_mem[MAX_RESERVED_REGIONS];

array中。

这里找出该entry。

```
struct reserved_mem {
    const char          *name;
    unsigned long           fdt_node;
    unsigned long           phandle;
    const struct reserved_mem_ops   *ops;
    phys_addr_t         base;
    phys_addr_t         size;
    void                *priv;
};
```

③

如果是dma-coherent.c处理该reserved memory，那么rmem->ops->device_init即指向 `rmem_dma_device_init` .

in drivers/base/dma-coherent.c

```
static int rmem_dma_device_init(struct reserved_mem *rmem, struct device *de
v)
{
    struct dma_coherent_mem *mem = rmem->priv;

    if (!mem &&
        dma_init_coherent_memory(rmem->base, rmem->base, rmem->size,
                     DMA_MEMORY_MAP | DMA_MEMORY_EXCLUSIVE,
                     &mem) != DMA_MEMORY_MAP) {
        pr_err("Reserved memory: failed to init DMA memory pool at %pa, size
 %ld MiB\n",
            &rmem->base, (unsigned long)rmem->size / SZ_1M);
        return -ENODEV;
    }
    rmem->priv = mem;
    dma_assign_coherent_memory(dev, mem);
    return 0;
}
```

由于rmem中的信息来自dtb中的node，所以只是physical address信息，所以需要翻译成kernel能使用的virtual information。

```
1.          dma_init_coherent_memory(rmem->base, rmem->base, rmem->size,
2.                       DMA_MEMORY_MAP | DMA_MEMORY_EXCLUSIVE,
3.                       &mem) != DMA_MEMORY_MAP) {
```

rmem->base is physical address of the reserved memory for the specific device
rmem->base is bus address of the reserved memory for the specific device
由于在ARM架构下，physical address与bus address是相同的，所以这里都是传递rmem->base
rmem->size is the size of the reserved memory for the specific device

```
1.  static int dma_init_coherent_memory(phys_addr_t phys_addr, dma_addr_t device
    _addr,
2.                  size_t size, int flags,
3.                  struct dma_coherent_mem **mem)
4.  {
5.      struct dma_coherent_mem *dma_mem = NULL;
6.      void __iomem *mem_base = NULL;
7.      int pages = size >> PAGE_SHIFT;
8.      int bitmap_size = BITS_TO_LONGS(pages) * sizeof(long);
9.
10.     if ((flags & (DMA_MEMORY_MAP | DMA_MEMORY_IO)) == 0)
11.         goto out;
12.     if (!size)
13.         goto out;
14.
15.     mem_base = ioremap(phys_addr, size);            ①
16.     if (!mem_base)
17.         goto out;
18.
19.     dma_mem = kzalloc(sizeof(struct dma_coherent_mem), GFP_KERNEL);    ②
20. if (!dma_mem)
21.         goto out;
22.     dma_mem->bitmap = kzalloc(bitmap_size, GFP_KERNEL);    ③
23.     if (!dma_mem->bitmap)
24.         goto out;
25.
26.     dma_mem->virt_base = mem_base;                  ④
27.     dma_mem->device_base = device_addr;             ⑤
28.     dma_mem->pfn_base = PFN_DOWN(phys_addr);        ⑥
29.     dma_mem->size = pages;                          ⑦
30.     dma_mem->flags = flags;
31.     spin_lock_init(&dma_mem->spinlock);
32.
33.     *mem = dma_mem;
34.
35.     if (flags & DMA_MEMORY_MAP)
36.         return DMA_MEMORY_MAP;
37.
38.     return DMA_MEMORY_IO;
39.
40. out:
41.     kfree(dma_mem);
42.     if (mem_base)
43.         iounmap(mem_base);
44.     return 0;
45. }
```

①

把reserved memory map to virtual address,毕竟kernel只能访问virtual address

②

```
struct dma_coherent_mem {
    void        *virt_base;
    dma_addr_t  device_base;
    unsigned long   pfn_base;
    int     size;
    int     flags;
    unsigned long   *bitmap;
    spinlock_t  spinlock;
};
```

管理per-device dma memory的structure

③

管理reserved memory的算法是有lib/bitmap.c下的函数实现的。

其本身也需要一定的space

④

reserved memory对应的virtual address

⑤

bus address,也是physical address,dma controller需要它

⑥

bus address对应的page frame number

⑦

dma_mem->size记录的是page number

在通过调用dma_init_coherent_memory()初始化好管理reserved dma memory的structure dma_coherent_mem以后

> dma_assign_coherent_memory(dev, mem);

```
static int dma_assign_coherent_memory(struct device *dev,
                      struct dma_coherent_mem *mem)
{
    if (dev->dma_mem)
        return -EBUSY;

    dev->dma_mem = mem;
    /* FIXME: this routine just ignores DMA_MEMORY_INCLUDES_CHILDREN */

    return 0;
}
```

即把专属于该device的dma memory的mem assign给 `dev->dma_mem` 。这样就建立了reserved dma memory与specific device之间的关系。

Question: device是怎样利用reserved dma memory的呢？

step 1:

device driver通过dma_alloc_coherent()来alloc `coherent` dma memory.

in arch/arm/inlcude/asm/dma-mapping.h

```
#define dma_alloc_coherent(d, s, h, f) dma_alloc_attrs(d, s, h, f, NULL)
```

```
static inline void *dma_alloc_attrs(struct device *dev, size_t size,
                      dma_addr_t *dma_handle, gfp_t flag,
                      struct dma_attrs *attrs)
{
struct dma_map_ops *ops = get_dma_ops(dev);
    void *cpu_addr;
    BUG_ON(!ops);

    cpu_addr = ops->alloc(dev, size, dma_handle, flag, attrs);
    debug_dma_alloc_coherent(dev, size, *dma_handle, cpu_addr);
    return cpu_addr;
}
```

step 2:

> get_dma_ops(dev);

```
static inline struct dma_map_ops *get_dma_ops(struct device *dev)
{
    if (xen_initial_domain())
        return xen_dma_ops;
    else
        return __generic_dma_ops(dev);
}
```

```
static inline struct dma_map_ops *__generic_dma_ops(struct device *dev)
{
    if (dev && dev->archdata.dma_ops)
        return dev->archdata.dma_ops;
    return &arm_dma_ops;
}
```

返回&arm_dma_ops
也就是

> struct dma_map_ops *ops = get_dma_ops(dev);

这里ops = &arm_dma_ops;

step 3:

in arch/arm/mm/dma-mapping.c

```
struct dma_map_ops arm_dma_ops = {
    .alloc          = arm_dma_alloc,
    .free           = arm_dma_free,
    .mmap           = arm_dma_mmap,
    .get_sgtable        = arm_dma_get_sgtable,
    .map_page       = arm_dma_map_page,
    .unmap_page     = arm_dma_unmap_page,
    .map_sg         = arm_dma_map_sg,
    .unmap_sg       = arm_dma_unmap_sg,
    .sync_single_for_cpu    = arm_dma_sync_single_for_cpu,
    .sync_single_for_device = arm_dma_sync_single_for_device,
    .sync_sg_for_cpu    = arm_dma_sync_sg_for_cpu,
    .sync_sg_for_device = arm_dma_sync_sg_for_device,
    .set_dma_mask       = arm_dma_set_mask,
};
EXPORT_SYMBOL(arm_dma_ops);
```

> cpu_addr = ops->alloc(dev, size, dma_handle, flag, attrs);

即cpu_addr = arm_dma_alloc(dev, size, dma_handle, flag, attrs);

```
/*
 * Allocate DMA-coherent memory space and return both the kernel remapped
 * virtual and bus address for that space.
 */
void *arm_dma_alloc(struct device *dev, size_t size, dma_addr_t *handle,
            gfp_t gfp, struct dma_attrs *attrs)
{
    pgprot_t prot = __get_dma_pgprot(attrs, PAGE_KERNEL);
    void *memory;

    if (dma_alloc_from_coherent(dev, size, handle, &memory))
        return memory;

    return __dma_alloc(dev, size, handle, gfp, prot, false,
                __builtin_return_address(0));
}
```

dma_alloc_from_coherent() —> **进入dma-coherent.c了！**

```
/**
 * dma_alloc_from_coherent() - try to allocate memory from the per-device coherent area
 *
 * @dev:    device from which we allocate memory
 * @size:   size of requested memory area
 * @dma_handle: This will be filled with the correct dma handle
 * @ret:    This pointer will be filled with the virtual address
 *      to allocated area.
 *
 * This function should be only called from per-arch dma_alloc_coherent()
 * to support allocation from per-device coherent memory pools.
 *
 * Returns 0 if dma_alloc_coherent should continue with allocating from
 * generic memory areas, or !0 if dma_alloc_coherent should return @ret.
 */
int dma_alloc_from_coherent(struct device *dev, ssize_t size,
                        dma_addr_t *dma_handle, void **ret)
{
    struct dma_coherent_mem *mem;
    int order = get_order(size);
    unsigned long flags;
    int pageno;

    if (!dev)
        return 0;
    mem = dev->dma_mem;     ①
    if (!mem)
        return 0;

    *ret = NULL;
    spin_lock_irqsave(&mem->spinlock, flags);

    if (unlikely(size > (mem->size << PAGE_SHIFT)))     ②
        goto err;

    pageno = bitmap_find_free_region(mem->bitmap, mem->size, order);     ③
    if (unlikely(pageno < 0))
        goto err;

    /*
     * Memory was found in the per-device area.
     */
    *dma_handle = mem->device_base + (pageno << PAGE_SHIFT);     ④
    *ret = mem->virt_base + (pageno << PAGE_SHIFT);              ⑤
    memset(*ret, 0, size);                                       ⑥
    spin_unlock_irqrestore(&mem->spinlock, flags);

    return 1;

err:
    spin_unlock_irqrestore(&mem->spinlock, flags);
    /*
```

```
     * In the case where the allocation can not be satisfied from the
     * per-device area, try to fall back to generic memory if the
     * constraints allow it.
     */
    return mem->flags & DMA_MEMORY_EXCLUSIVE;
}
EXPORT_SYMBOL(dma_alloc_from_coherent);
```

①
获得该devcie专属的dma memory的 `handle`
②
申请分配的size是否大于该reserved dma memory
③
查询free space，并返回起始page number
④
*dma_handle返回分配的dma space的bus address
⑤
*ret返回分配的dma space的virtual address
⑥
返回的dma space是清零的

---

dma-coherent.c的总结如下：
如果device有专属于自己的dma memory,那么在该device driver中通过 `dma_alloc_coherent()` 分配dma memory时首先会从该专属dma memory中分配；如果没有专属的dma memory或者从专属的dma memory分配失败，才会通过generic dma allocator分配。

```
void *arm_dma_alloc(struct device *dev, size_t size, dma_addr_t *handle,
            gfp_t gfp, struct dma_attrs *attrs)
{
    pgprot_t prot = __get_dma_pgprot(attrs, PAGE_KERNEL);
    void *memory;

    if (dma_alloc_from_coherent(dev, size, handle, &memory))
        return memory;

    return __dma_alloc(dev, size, handle, gfp, prot, false,
                __builtin_return_address(0));
}
```

__dma_alloc()就是generic allocator.