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);
```

```
struct dma_map_ops arm_coherent_dma_ops = {
    .alloc          = arm_coherent_dma_alloc,
    .free           = arm_coherent_dma_free,
    .mmap           = arm_dma_mmap,
    .get_sgtable        = arm_dma_get_sgtable,
    .map_page       = arm_coherent_dma_map_page,
    .map_sg         = arm_dma_map_sg,
    .set_dma_mask       = arm_dma_set_mask,
};
EXPORT_SYMBOL(arm_coherent_dma_ops);
```

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

```
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;
}

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);
}
```

dma APIs 基本上都最终会通过get_dma_ops() route到上面两个global variables中的某个callback中。

暴露给driver developer的dma_alloc_coherent() / dma_free_coherent()

```
#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;
}
```

```
#define dma_free_coherent(d, s, c, h) dma_free_attrs(d, s, c, h, NULL)

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

    debug_dma_free_coherent(dev, size, cpu_addr, dma_handle);
    ops->free(dev, size, cpu_addr, dma_handle, attrs);
}
```

最终调用的是这里的ops->alloc() / ops->free() callback.

而从__generic_dma_ops()看

```
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;
}
```

如果dev->archdata.dma_ops存在则先使用之。

```
static inline void set_dma_ops(struct device *dev, struct dma_map_ops *ops)
{
    BUG_ON(!dev);
    dev->archdata.dma_ops = ops;
}
```

```
static inline int set_arch_dma_coherent_ops(struct device *dev)
{
    set_dma_ops(dev, &arm_coherent_dma_ops);
    return 0;
}
```

即developer可以通过set_arch_dma_coherent_ops()来定制dma operations.

in drivers/of/platform.c

```
static void of_dma_configure(struct device *dev)
{
    u64 dma_addr, paddr, size;
    int ret;

    /*
     * Set default dma-mask to 32 bit. Drivers are expected to setup
     * the correct supported dma_mask.
     */
    dev->coherent_dma_mask = DMA_BIT_MASK(32);

    /*
     * Set it to coherent_dma_mask by default if the architecture
     * code has not set it.
     */
    if (!dev->dma_mask)
        dev->dma_mask = &dev->coherent_dma_mask;

    /*
     * if dma-coherent property exist, call arch hook to setup
     * dma coherent operations.
     */
    if (of_dma_is_coherent(dev->of_node)) {        ①
        set_arch_dma_coherent_ops(dev);                ②
        dev_dbg(dev, "device is dma coherent\n");
    }

    /*
     * if dma-ranges property doesn't exist - just return else
     * setup the dma offset
     */
    ret = of_dma_get_range(dev->of_node, &dma_addr, &paddr, &size);
    if (ret < 0) {
        dev_dbg(dev, "no dma range information to setup\n");
        return;
    }

    /* DMA ranges found. Calculate and set dma_pfn_offset */
    dev->dma_pfn_offset = PFN_DOWN(paddr - dma_addr);
    dev_dbg(dev, "dma_pfn_offset(%#08lx)\n", dev->dma_pfn_offset);
}
```

①

```
/**
 * of_dma_is_coherent - Check if device is coherent
 * @np: device node
 *
 * It returns true if "dma-coherent" property was found
 * for this device in DT.
 */
bool of_dma_is_coherent(struct device_node *np)
{
    struct device_node *node = of_node_get(np);

    while (node) {
        if (of_property_read_bool(node, "dma-coherent")) {    (A)
            of_node_put(node);
            return true;
        }
        node = of_get_next_parent(node);
    }
    of_node_put(node);
    return false;
}
```

(A)检查在dtb的device node描述中是否有如下property

> dma-coherent;

②

设置该device的dma operation为 `arm_coherent_dma_ops`

目前在pegmatite SoC LSP中也应用到了。

in arch/arm/mach-pegmatite/pegmatite.c

```
static int dma_notifier(struct notifier_block *nb,
            unsigned long event, void *__dev)
{
    struct device *dev = __dev;

    if (event != BUS_NOTIFY_ADD_DEVICE)
        return NOTIFY_DONE;
    set_dma_ops(dev, &pegmatite_dma_ops);            (I)

    if (strstr(dev_name(dev), "d0700000.stmmac")) {
         if (dev->dma_mask)
             *dev->dma_mask = DMA_BIT_MASK(31);
         dev->coherent_dma_mask = DMA_BIT_MASK(31);
    }

    return NOTIFY_OK;
}
```

而这里的pegmatite_dma_ops为

```
static struct dma_map_ops pegmatite_dma_ops;

static void init_dma_ops (void)
{
    memcpy(&pegmatite_dma_ops, &arm_dma_ops, sizeof(pegmatite_dma_ops));
    pegmatite_dma_ops.map_page = pegmatite_map_page;
    pegmatite_dma_ops.map_sg = pegmatite_map_sg;
}
```

即pegmatite_dma_ops修改了原来arm_dma_ops中的
.map_page and .map_sg 2 callbacks.

这样dma_map_sg() and dma_map_page()两个 dma operation APIs被定制化了！

```
#define dma_map_sg(d, s, n, r) dma_map_sg_attrs(d, s, n, r, NULL)
```

```
static inline int dma_map_sg_attrs(struct device *dev, struct scatterlist *s
g,
                   int nents, enum dma_data_direction dir,
                   struct dma_attrs *attrs)
{
    struct dma_map_ops *ops = get_dma_ops(dev);     ①
int i, ents;
    struct scatterlist *s;

    for_each_sg(sg, s, nents, i)
        kmemcheck_mark_initialized(sg_virt(s), s->length);
    BUG_ON(!valid_dma_direction(dir));
    ents = ops->map_sg(dev, sg, nents, dir, attrs);     ②
    debug_dma_map_sg(dev, sg, nents, ents, dir);

    return ents;
}
```

```
static inline dma_addr_t dma_map_page(struct device *dev, struct page *page,
                     size_t offset, size_t size,
                     enum dma_data_direction dir)
{
    struct dma_map_ops *ops = get_dma_ops(dev);
    dma_addr_t addr;

    kmemcheck_mark_initialized(page_address(page) + offset, size);
    BUG_ON(!valid_dma_direction(dir));
    addr = ops->map_page(dev, page, offset, size, dir, NULL);
    debug_dma_map_page(dev, page, offset, size, dir, addr, false);

    return addr;
}
```

①
由于(I)处的code

> set_dma_ops(dev, &pegmatite_dma_ops);

所以这里ops = &pegmatite_dma_ops
②

> ents = ops->map_sg(dev, sg, nents, dir, attrs);

等价于

> ents = pegmatite_map_sg(dev, sg, nents, dir, attrs);

---

pegmatite SoC对dma operation定制化

in arch/arm/mach-pegmatite/pegmatite.c

```
1.  /* These  DMA functions should behave the same as the generic ARM
2.   * functions, but with an additional sanity check to verify that we
3.   * don't try to DMA to DRAM above 2GB, because HW can't address that
4.   * memory.  Unfortunately the implementation for arm_dma_map_page is
5.   * static, so we can't just call them. */
6.  static dma_addr_t pegmatite_map_page(struct device *dev, struct page *page,
7.                       unsigned long offset, size_t size,
8.                       enum dma_data_direction dir,
9.                       struct dma_attrs *attrs)
10. {
11.     dma_addr_t addr = pfn_to_dma(dev, page_to_pfn(page)) + offset;
12.     BUG_ON(page_to_phys(page) >= SZ_2G);                ①
13.     dma_sync_single_for_device(dev, addr, size, dir);       ②
14.     return addr;
15. }
16.
17. static int pegmatite_map_sg(struct device *dev, struct scatterlist *sg, int
    nents,
18.                 enum dma_data_direction dir, struct dma_attrs *attrs)
19. {
20.     int i;
21.     for (i=0; i<nents; i++)
22.         BUG_ON(page_to_phys(sg_page(&sg[i])) >= SZ_2G);     ③
23.     return arm_dma_map_sg(dev, sg, nents, dir, attrs);    ④
24. }
```

①②③④
定制化的本意只是为了check这两个dma memory alloction的function接受到的物理地址是否 < 2G(因为在pegmatite SoC上DMA不能寻址2G以上物理地址)。