当device / driver加入到bus上时，bus会发出对应的notification,kernel code可以关注这些notification，并对device或driver进行修改！

in linux/device.h

```
/* All 4 notifers below get called with the target struct device *
 * as an argument. Note that those functions are likely to be called
 * with the device lock held in the core, so be careful.
 */
#define BUS_NOTIFY_ADD_DEVICE        0x00000001 /* device added */
#define BUS_NOTIFY_DEL_DEVICE        0x00000002 /* device to be removed */
#define BUS_NOTIFY_REMOVED_DEVICE    0x00000003 /* device removed */
#define BUS_NOTIFY_BIND_DRIVER       0x00000004 /* driver about to be
                                bound */
#define BUS_NOTIFY_BOUND_DRIVER      0x00000005 /* driver bound to device */
#define BUS_NOTIFY_UNBIND_DRIVER     0x00000006 /* driver about to be
                                unbound */
#define BUS_NOTIFY_UNBOUND_DRIVER    0x00000007 /* driver is unbound
                                from the device */
```

注册关注这些notification的APIs

```
/*
 * Bus notifiers: Get notified of addition/removal of devices
 * and binding/unbinding of drivers to devices.
 * In the long run, it should be a replacement for the platform
 * notify hooks.
 */
struct notifier_block;

extern int bus_register_notifier(struct bus_type *bus,
                 struct notifier_block *nb);
extern int bus_unregister_notifier(struct bus_type *bus,
                   struct notifier_block *nb);
```

kernel 4.2.8中pegmatite SoC的 `drivers/platform/pegmatite/dma.c` 就利用了这种机制来实现对dma memory分配的physical address是否大于2G进行检查。

```c
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/io.h>
#include <linux/dma-mapping.h>
#include <linux/sizes.h>
#include <linux/pci.h>
#include <linux/platform_device.h>

static struct dma_map_ops pegmatite_dma_ops;
static struct dma_map_ops *orig_dma_ops;
static u64 default_dmamask = DMA_BIT_MASK(31);

struct platform_device dummy_default_dma_dev = {
    .name   = "dummy_default_dma",
    .id = -1,
    .dev    = {
        .dma_mask       = &default_dmamask,
        .coherent_dma_mask  = DMA_BIT_MASK(31),
    },
};

static void *pegmatite_alloc(struct device *dev, size_t size,
                dma_addr_t *dma_handle, gfp_t gfp,
                struct dma_attrs *attrs)
{
    if (WARN_ON_ONCE(!dev)) {
        dev = &dummy_default_dma_dev.dev;
    }
    return orig_dma_ops->alloc(dev, size, dma_handle, gfp, attrs);
}

static void pegmatite_free(struct device *dev, size_t size,
               void *vaddr, dma_addr_t dma_handle,
               struct dma_attrs *attrs)
{
    if (WARN_ON_ONCE(!dev)) {
        dev = &dummy_default_dma_dev.dev;
    }
    orig_dma_ops->free(dev, size, vaddr, dma_handle, attrs);
}

/* These  DMA functions should behave the same as the generic ARM
 * functions, but with an additional sanity check to verify that we
 * don't try to DMA to DRAM above 2GB, because HW can't address that
 * memory. */
static dma_addr_t pegmatite_map_page(struct device *dev, struct page *page,
                     unsigned long offset, size_t size,
                     enum dma_data_direction dir,
                     struct dma_attrs *attrs)
{
    BUG_ON(page_to_phys(page) >= SZ_2G);    ⑥
    if (WARN_ON_ONCE(!dev)) {
        dev = &dummy_default_dma_dev.dev;
```

```
54.     }
55.     return orig_dma_ops->map_page(dev, page, offset, size, dir, attrs);
56. }
57. 
58. static int pegmatite_map_sg(struct device *dev, struct scatterlist *sg, int
    nents,
59.                 enum dma_data_direction dir, struct dma_attrs *attrs)
60. {
61.     int i;
62.     struct scatterlist *s;
63. 
64.     for_each_sg(sg, s, nents, i) {
65.         BUG_ON(page_to_phys(sg_page(s)) >= SZ_2G);
66.     }
67. 
68.     if (WARN_ON_ONCE(!dev)) {
69.         dev = &dummy_default_dma_dev.dev;
70.     }
71.     return orig_dma_ops->map_sg(dev, sg, nents, dir, attrs);
72. }
73. 
74. static void init_dma_ops (void)
75. {
76.     int rc;
77. 
78.     orig_dma_ops = get_dma_ops(NULL);
79.     memcpy(&pegmatite_dma_ops, orig_dma_ops, sizeof(pegmatite_dma_ops));
80.     pegmatite_dma_ops.alloc = pegmatite_alloc;
81.     pegmatite_dma_ops.free = pegmatite_free;
82.     pegmatite_dma_ops.map_page = pegmatite_map_page;
83.     pegmatite_dma_ops.map_sg = pegmatite_map_sg;
84. #ifdef CONFIG_ARM64
85.     /* Set the global default dma ops */
86.     dma_ops = &pegmatite_dma_ops;
87. #endif
88.     rc = platform_device_register(&dummy_default_dma_dev);
89.     if (rc) {
90.         pr_err("Registering dma device failed! %d\n", rc);
91.         return;
92.     }
93. }
94. 
95. static int dma_notifier(struct notifier_block *nb,
96.             unsigned long event, void *__dev)
97. {
98.     struct device *dev = __dev;
99. 
100.     if (event != BUS_NOTIFY_BIND_DRIVER)        ③
101.         return NOTIFY_DONE;
102.     dev->archdata.dma_ops = &pegmatite_dma_ops;    ④
103. 
104.     if (strstr(dev_name(dev), "d0700000.stmmac")) {    ⑤
105.          if (dev->dma_mask)
106.              *dev->dma_mask = DMA_BIT_MASK(31);
```

```
107.            dev->coherent_dma_mask = DMA_BIT_MASK(31);
108.        }
109.
110.        return NOTIFY_OK;
111.    }
112.
113.    static struct notifier_block dma_nb = {
114.        .notifier_call = dma_notifier,
115.    };
116.
117.    static int __init dma_init(void)
118.    {
119.        init_dma_ops();
120.        bus_register_notifier(&platform_bus_type,     ①
121.                              &dma_nb);
122.    #if IS_ENABLED(CONFIG_PCI)
123.        bus_register_notifier(&pci_bus_type,          ②
124.                              &dma_nb);
125.    #endif
126.        return 0;
127.    }
128.    arch_initcall(dma_init);
```

①②

关注 `platform` bus和 `pci` bus上的driver。

③

只关心 `BUS_NOTIFY_BIND_DRIVER` notification，即当driver与对应的device将要bind(绑定)之时。

④

替换原来的 `struct dma_map_ops`

⑤

可以针对特定的device

⑥

在map page以前做针对pegmatite SoC的检查。