**clock的初始化**

in drivers/clk/pegmatite/clkgate.c

> CLK_OF_DECLARE(pegmatite_clkgate, "marvell,pegmatite-clkgate",
> of_pegmatite_clkgate_setup);

in include/linux/clk-provider.h

```
#define CLK_OF_DECLARE(name, compat, fn) OF_DECLARE_1(clk, name, compat, fn)
```

in include/of.h

```
#define OF_DECLARE_1(table, name, compat, fn) \
        _OF_DECLARE(table, name, compat, fn, of_init_fn_1)


#define _OF_DECLARE(table, name, compat, fn, fn_type)           \
    static const struct of_device_id __of_table_##name      \
        __used __section(__##table##_of_table)          \
         = { .compatible = compat,              \
             .data = (fn == (fn_type)NULL) ? fn : fn  }
```

也就是定义了如下code

```
static const struct of_device_id __of_table_pegmatite_clkgate
        __used __section(__clk_of_table)
         = { .compatible = "marvell,pegmatite-clkgate",
             .data = (of_pegmatite_clkgate_setup == (of_init_fn_1)NULL) ? of
_pegmatite_clkgate_setup : of_pegmatite_clkgate_setup }
```

> typedef void (*of_init_fn_1)(struct device_node *);
>
> static void __init of_pegmatite_clkgate_setup(struct device_node *node)

of_pegmatite_clkgate_setup是of_init_fn_1 type的function pointer。

上面的macro也就是定义了一个struct of_device_id的variable

```
/*
 * Struct used for matching a device
 */
struct of_device_id
{
    char    name[32];
    char    type[32];
    char    compatible[128];
    const void *data;
};

struct of_device_id __of_table_pegmatite_clkgate
{
    .compatible = "marvell,pegmatite-clkgate",
    .data = of_pegmatite_clkgate_setup,
};
```

该variable被放置与 `__section(__clk_of_table)` 中。

在vmlinux.S kernel链接脚本中

```
 .init.data : {
  *(.init.data) *(.meminit.data) *(.init.rodata) *(.meminit.rodata) . = ALIG
N(8); __clk_of_table = .; *(__clk_of_table) *(__clk_of_table_end) . = ALIGN(
8); __reservedmem_of_table = .; *(__reservedmem_of_table) *(__reservedmem_of
_table_end) . = ALIGN(8); __clksrc_of_table = .; *(__clksrc_of_table) *(__cl
ksrc_of_table_end) . = ALIGN(8); __cpu_method_of_table = .; *(__cpu_method_o
f_table) *(__cpu_method_of_table_end) . = ALIGN(32); __dtb_start = .; *(.dtb
.init.rodata) __dtb_end = .; . = ALIGN(8); __irqchip_of_table = .; *(__irqch
ip_of_table) *(__irqchip_of_table_end)
  . = ALIGN(16); __setup_start = .; *(.init.setup) __setup_end = .;
  __initcall_start = .; *(.initcallearly.init) __initcall0_start = .; *(.ini
tcall0.init) *(.initcall0s.init) __initcall1_start = .; *(.initcall1.init) *
(.initcall1s.init) __initcall2_start = .; *(.initcall2.init) *(.initcall2s.i
nit) __initcall3_start = .; *(.initcall3.init) *(.initcall3s.init) __initcal
l4_start = .; *(.initcall4.init) *(.initcall4s.init) __initcall5_start = .;
*(.initcall5.init) *(.initcall5s.init) __initcallrootfs_start = .; *(.initca
llrootfs.init) *(.initcallrootfss.init) __initcall6_start = .; *(.initcall6.
init) *(.initcall6s.init) __initcall7_start = .; *(.initcall7.init) *(.initc
all7s.init) __initcall_end = .;
  __con_initcall_start = .; *(.con_initcall.init) __con_initcall_end = .;
  __security_initcall_start = .; *(.security_initcall.init) __security_initc
all_end = .;
  . = ALIGN(4); __initramfs_start = .; *(.init.ramfs) . = ALIGN(8); *(.init.
ramfs.info)
 }
```

所有在section "__clk_of_table"中的variable都会被放置在一起，组成一个struct of_device_id array,用
__clk_of_table symbol来指向，效果就像下面的code。

```
struct of_device_id __clk_of_table[];
```

每一个 `CLK_OF_DECLARE macro` 就是该array的一个成员，并且都是被初始化过的。

其实在include/linux/clk-provider.h中就输出了该array的symbol。

> extern struct of_device_id __clk_of_table;

**kernel对整个clock的初始化**

in init/main.c/start_kernel()

> time_init();

in arch/arm/kernel/time.c

```
1.  void __init time_init(void)
2.  {
3.      if (machine_desc->init_time) {
4.          machine_desc->init_time();
5.      } else {
6.  #ifdef CONFIG_COMMON_CLK
7.          of_clk_init(NULL);
8.  #endif
9.          clocksource_of_init();
10.     }
11. }
```

in drivers/clk/clk.c

```c
/**
 * of_clk_init() - Scan and init clock providers from the DT
 * @matches: array of compatible values and init functions for providers.
 *
 * This function scans the device tree for matching clock providers
 * and calls their initialization functions. It also does it by trying
 * to follow the dependencies.
 */
void __init of_clk_init(const struct of_device_id *matches)
{
    const struct of_device_id *match;
    struct device_node *np;
    struct clock_provider *clk_provider, *next;
    bool is_init_done;
    bool force = false;

    if (!matches)
        matches = &__clk_of_table;

    /* First prepare the list of the clocks providers */
    for_each_matching_node_and_match(np, matches, &match) {     ①
        struct clock_provider *parent =
            kzalloc(sizeof(struct clock_provider),  GFP_KERNEL);

        parent->clk_init_cb = match->data;
        parent->np = np;
        list_add_tail(&parent->node, &clk_provider_list);
    }

    while (!list_empty(&clk_provider_list)) {                   ②
        is_init_done = false;
        list_for_each_entry_safe(clk_provider, next,
                    &clk_provider_list, node) {
            if (force || parent_ready(clk_provider->np)) {      ③

                clk_provider->clk_init_cb(clk_provider->np);    ④
                of_clk_set_defaults(clk_provider->np, true);    ⑤

                list_del(&clk_provider->node);
                kfree(clk_provider);
                is_init_done = true;
            }
        }

        /*
         * We didn't manage to initialize any of the
         * remaining providers during the last loop, so now we
         * initialize all the remaining ones unconditionally
         * in case the clock parent was not mandatory
         */
        if (!is_init_done)
            force = true;
    }
```

```
54.  }
```

由于clock之间有father-child的依赖关系，所以上面的code要处理之。father必然要先于child被初始化。

①
把__clk_of_table[]  array中的每个成员中的"compatible" string与device tree中的node的"compatible"
property比较，以找到对应的device node,然后链入clk_provider_list global list中。

> static LIST_HEAD(clk_provider_list);

enumerate the list可以获得所有当前系统注册的所有clock的信息。当然the list的存在是很短暂的。

②
enumerate clk_provider_list,处理一个node就删除之。

③
要处理父子之间的依赖关系

④
调用各个clock module提供的初始化函数，比如 `of_pegmatite_clkgate_setup()`
由于clock module的初始化比较早，所以显然并不是所有kernel function都能调用的。如果相应moudle的初始化在clock framework
之后，显然它输出的function就不能被调用。

⑤
最早对clock的设置是在这时候。

对应driver中对设备相关的clock的设定(类似如下code)其实是 `第二次` 的设定了。

```
1.      clk = clk_get(&pdev->dev, NULL);
2.      if(IS_ERR(clk))
3.      {
4.          err = PTR_ERR(clk);
5.          dev_dbg(&pdev->dev, "fail to get clock!\n");
6.          goto err_handling_3;
7.      }
8.
9.      if(clk_prepare_enable(clk))
10.     {
11.         dev_dbg(&pdev->dev, "fail to prepare clock!\n");
12.         clk_put(clk);
13.         goto err_handling_2;
14.     }
```