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

```
DT_MACHINE_START(PEGMATITE_DT, "Marvell Pegmatite (Device Tree)")
#ifdef CONFIG_SMP
        .smp            = smp_ops(pegmatite_smp_ops),
#endif
        .init_machine  = pegmatite_dt_init,
        .map_io         = pegmatite_map_io,
        .init_early = pegmatite_init_early,
        .init_irq    = pegmatite_init_irq,
        .init_time  = pegmatite_timer_and_clk_init,
        .restart    = pegmatite_restart,
        .dt_compat     = pegmatite_dt_compat,
#ifdef CONFIG_ZONE_DMA
        .dma_zone_size     = SZ_256M,
#endif
MACHINE_END
```

```
#define DT_MACHINE_START(_name, _namestr)               \
static const struct machine_desc __mach_desc_##_name\
 __used                                          \
 __attribute__((__section__(".arch.info.init"))) = {  \
        .nr          = ~0,                      \
        .name            = _namestr,
```

```
#endif



#define MACHINE_END                     \
};
```

实际上定义了

```
static const struct machine_desc __mach_desc_PEGMATITE_DT
__used __attribute__((__section__(".arch.info.init"))) = {
        .nr             = ~0,
        .name           = "Marvell Pegmatite (Device Tree)",
#ifdef CONFIG_SMP
        .smp            = smp_ops(pegmatite_smp_ops),
#endif
        .init_machine   = pegmatite_dt_init,
        .map_io         = pegmatite_map_io,
        .init_early     = pegmatite_init_early,
        .init_irq       = pegmatite_init_irq,
        .init_time      = pegmatite_timer_and_clk_init,
        .restart        = pegmatite_restart,
        .dt_compat      = pegmatite_dt_compat,
#ifdef CONFIG_ZONE_DMA
```

```
	.dma_zone_size	= SZ_256M,
#endif
};
```

machine_desc structure被放入".arch.info.init" section。

在vmlinux.lds.S中有

```
	.init.arch.info : {
		__arch_info_begin = .;
		*(.arch.info.init)
		__arch_info_end = .;
	}
```

即DT_MACHINE_START macro定义的machine_desc structure variable组成了一个array,该array的head为__arch_info_begin，tail为__arch_info_end.

in arch/arm/kernel/setup.c

```
void __init setup_arch(char **cmdline_p)
{
	const struct machine_desc *mdesc;

	setup_processor();
```

```
mdesc = setup_machine_fdt(__atags_pointer);
if (!mdesc)
        mdesc = setup_machine_tags(__atags_pointer, __machine_arch_type);
machine_desc = mdesc;
machine_name = mdesc->name;

if (mdesc->reboot_mode != REBOOT_HARD)
        reboot_mode = mdesc->reboot_mode;

init_mm.start_code = (unsigned long) _text;
init_mm.end_code   = (unsigned long) _etext;
init_mm.end_data   = (unsigned long) _edata;
init_mm.brk        = (unsigned long) _end;

/* populate cmd_line too for later use, preserving boot_command_line */
strlcpy(cmd_line, boot_command_line, COMMAND_LINE_SIZE);
*cmdline_p = cmd_line;

parse_early_param();

early_paging_init(mdesc, lookup_processor_type(read_cpuid_id()));
setup_dma_zone(mdesc);
sanity_check_meminfo();
arm_memblock_init(mdesc);
```

```
	paging_init(mdesc);
	request_standard_resources(mdesc);

	if (mdesc->restart)
		arm_pm_restart = mdesc->restart;

	unflatten_device_tree();

	arm_dt_init_cpu_maps();
	psci_init();
#ifdef CONFIG_SMP
	if (is_smp()) {
		if (!mdesc->smp_init || !mdesc->smp_init()) {
			if (psci_smp_available())
				smp_set_ops(&psci_smp_ops);
			else if (mdesc->smp)
				smp_set_ops(mdesc->smp);
		}
		smp_init_cpus();
		smp_build_mpidr_hash();
	}
#endif

	if (!is_smp())
		hyp_mode_check();
```

```
	reserve_crashkernel();

#ifdef CONFIG_MULTI_IRQ_HANDLER
	handle_arch_irq = mdesc->handle_irq;
#endif

#ifdef CONFIG_VT
#if defined(CONFIG_VGA_CONSOLE)
	conswitchp = &vga_con;
#elif defined(CONFIG_DUMMY_CONSOLE)
	conswitchp = &dummy_con;
#endif
#endif

	if (mdesc->init_early)
		mdesc->init_early();
}


mdesc = setup_machine_fdt(__atags_pointer);

get machine_desc according to dtb.

/**
```

```
 * setup_machine_fdt - Machine setup when an dtb was passed to the kernel
 * @dt_phys: physical address of dt blob
 *
 * If a dtb was passed to the kernel in r2, then use it to choose the
 * correct machine_desc and to setup the system.
 */
const struct machine_desc * __init setup_machine_fdt(unsigned int dt_phys)
{
	const struct machine_desc *mdesc, *mdesc_best = NULL;

#ifdef CONFIG_ARCH_MULTIPLATFORM                                    (1)
	DT_MACHINE_START(GENERIC_DT, "Generic DT based system")
	MACHINE_END

	mdesc_best = &__mach_desc_GENERIC_DT;
#endif

	if (!dt_phys || !early_init_dt_verify(phys_to_virt(dt_phys)))
		return NULL;

	mdesc = of_flat_dt_match_machine(mdesc_best, arch_get_next_mach);  (2)

	if (!mdesc) {
		const char *prop;
		int size;
```

```
        unsigned long dt_root;


        early_print("\nError: unrecognized/unsupported "
                    "device tree compatible list:\n[ ");


        dt_root = of_get_flat_dt_root();
        prop = of_get_flat_dt_prop(dt_root, "compatible", &size);
        while (size > 0) {
                early_print("'%s' ", prop);
                size -= strlen(prop) + 1;
                prop += strlen(prop) + 1;
        }
        early_print("]\n\n");


        dump_machine_table(); /* does not return */
}


/* We really don't want to do this, but sometimes firmware provides buggy data */
if (mdesc->dt_fixup)
        mdesc->dt_fixup();


early_init_dt_scan_nodes();


/* Change machine number to match the mdesc we're using */
__machine_arch_type = mdesc->nr;
```

```
        return mdesc;
}
```

(1)

CONFIG_ARCH_MULTIPLATFORM=y

所以在__arch_info_begin and __arch_info_end之间由两个machine_desc variable。

__mach_desc_GENERIC_DT

__mach_desc_PEGMATITE_DT

(2)

```
static const void * __init arch_get_next_mach(const char *const **match)
{
        static const struct machine_desc *mdesc = __arch_info_begin;
        const struct machine_desc *m = mdesc;

        if (m >= __arch_info_end)
                return NULL;

        mdesc++;
        *match = m->dt_compat;
        return m;
}
```

依次return __arch_info_begin and __arch_info_end之间的machine_desc variable.

of_flat_dt_match_machine() function用dtb中的root的“compatible” property与枚举获得的machine_desc中的dt_compat field比较，如果match，则该dtb就是描述了该machine的信息。

```
struct machine_desc {
        unsigned int            nr;             /* architecture number  */
        const char              *name;          /* architecture name    */
        unsigned long           atag_offset;    /* tagged list (relative) */
        const char *const       *dt_compat;     /* array of device tree
                                                 * 'compatible' strings      */

        unsigned int            nr_irqs;        /* number of IRQs */

#ifdef CONFIG_ZONE_DMA
        phys_addr_t             dma_zone_size;  /* size of DMA-able area */
#endif

        unsigned int            video_start;    /* start of video RAM   */
        unsigned int            video_end;      /* end of video RAM*/

        unsigned char           reserve_lp0 :1;     /* never has lp0    */
        unsigned char           reserve_lp1 :1;     /* never has lp1    */
        unsigned char           reserve_lp2 :1;     /* never has lp2    */
```

```
        enum reboot_modereboot_mode; /* default restart mode    */
        unsigned        l2c_aux_val;    /* L2 cache aux value     */
        unsigned        l2c_aux_mask;       /* L2 cache aux mask    */
        void            (*l2c_write_sec)(unsigned long, unsigned);
        struct smp_operations   *smp;           /* SMP operations   */
        bool            (*smp_init)(void);
        void            (*fixup)(struct tag *, char **);
        void            (*dt_fixup)(void);
        void            (*init_meminfo)(void);
        void            (*reserve)(void);/* reserve mem blocks   */
        void            (*map_io)(void);/* IO mapping function     */
        void            (*init_early)(void);
        void            (*init_irq)(void);
        void            (*init_time)(void);
        void            (*init_machine)(void);
        void            (*init_late)(void);
#ifdef CONFIG_MULTI_IRQ_HANDLER
        void            (*handle_irq)(struct pt_regs *);
#endif
        void            (*restart)(enum reboot_mode, const char *);
};
```

struct machine_desc中的function pointer variable就像design pattern中的Template Method Pattern中的"method",提供了developer定制kernel与arch相关部分的启动过程。

下面是这些function pointer被调用的时机。

1. smp_init

in arch/arm/kernel/setup.c, setup_arch()

```
#ifdef CONFIG_SMP
	if (is_smp()) {
		if (!mdesc->smp_init || !mdesc->smp_init()) {
			if (psci_smp_available())
				smp_set_ops(&psci_smp_ops);
			else if (mdesc->smp)
				smp_set_ops(mdesc->smp);
		}
		smp_init_cpus();
		smp_build_mpidr_hash();
	}
#endif
```

2. fixup

在enable device tree arch(OF)的情况下，setup_machine_tags()不会被调用到，可以忽略。

in arch/arm/kernel/atags.c, setup_machine_tags()

```
        if (__atags_pointer)
                tags = phys_to_virt(__atags_pointer);
        else if (mdesc->atag_offset)
                tags = (void *)(PAGE_OFFSET + mdesc->atag_offset);

#if defined(CONFIG_DEPRECATED_PARAM_STRUCT)
        /*
         * If we have the old style parameters, convert them to
         * a tag list.
         */
        if (tags->hdr.tag != ATAG_CORE)
                convert_to_tag_list(tags);
#endif
        if (tags->hdr.tag != ATAG_CORE) {
                early_print("Warning: Neither atags nor dtb found\n");
                tags = (struct tag *)&default_tags;
        }

        if (mdesc->fixup)
                mdesc->fixup(tags, &from);

        if (tags->hdr.tag == ATAG_CORE) {
                if (memblock_phys_mem_size())
                        squash_mem_tags(tags);
```

```
            save_atags(tags);
            parse_tags(tags);
        }
```

3. dt_fixup

in arch/arm/kernel/devtree.c, setup_machine_fdt()

```
        /* We really don't want to do this, but sometimes firmware provides buggy data */
        if (mdesc->dt_fixup)
            mdesc->dt_fixup();
```

4. init_meminfo

in arch/arm/mm/mmu.c, early_paging_init()

```
void __init early_paging_init(const struct machine_desc *mdesc,
                              struct proc_info_list *procinfo)
{
        if (mdesc->init_meminfo)
            mdesc->init_meminfo();
}
```

5. reserve (reserve mem blocks)

in arch/arm/mm/init.c, arm_memblock_init()

```
    /* reserve any platform specific memblock areas */
    if (mdesc->reserve)
        mdesc->reserve();
```

6. map_io (IO mapping function)

in arch/arm/mm/mmu.c, devicemaps_init()

```
    /*
     * Ask the machine support to map in the statically mapped devices.
     */
    if (mdesc->map_io)
        mdesc->map_io();
    else
        debug_ll_io_init();
```

7. init_early

in arch/arm/kernel/setup.c, setup_arch()

```
    if (mdesc->init_early)
        mdesc->init_early();
```

8. init_irq (在deisable device tree arch的情况下被调用)

in arch/arm/kernel/irq.c, init_IRQ()

```
    if (IS_ENABLED(CONFIG_OF) && !machine_desc->init_irq)
        irqchip_init();
    else
        machine_desc->init_irq();
```

9. init_time

in arch/arm/time.c, time_init()

```
void __init time_init(void)
{
    if (machine_desc->init_time) {
        machine_desc->init_time();
    } else {
#ifdef CONFIG_COMMON_CLK
        of_clk_init(NULL);
```

```
#endif
        clocksource_of_init();
    }
}
```

10. init_machine

in arch/arm/kernel/setup.c, customize_machine()

```
static int __init customize_machine(void)
{
    /*
     * customizes platform devices, or adds new ones
     * On DT based machines, we fall back to populating the
     * machine from the device tree, if no callback is provided,
     * otherwise we would always need an init_machine callback.
     */
    if (machine_desc->init_machine)
        machine_desc->init_machine();
#ifdef CONFIG_OF
    else
        of_platform_populate(NULL, of_default_bus_match_table,
                             NULL, NULL);
#endif
    return 0;
```

```
}
arch_initcall(customize_machine);
```

11. init_late

in arch/arm/kernel/setup.c, init_machine_late

```
static int __init init_machine_late(void)
{
	if (machine_desc->init_late)
		machine_desc->init_late();
	return 0;
}
late_initcall(init_machine_late);
```

12. handle_irq

in arch/arm/kernel/setup.c, setup_arch()

```
#ifdef CONFIG_MULTI_IRQ_HANDLER
	handle_arch_irq = mdesc->handle_irq;
#endif
```

而handle_arch_irq则在arch/arm/kernel/entry-armv.S中

```
#ifdef CONFIG_MULTI_IRQ_HANDLER
	.globl	handle_arch_irq
handle_arch_irq:
	.space	4
#endif
```

即在machine_desc中可以直接定义处理hardware interrupt的handler。如果定义，则当发生hardware interrupt时，就直接跳转到该handler中。

在Gr2 / Gs2 LSP中并没有直接定义，而是有GIC负责fill handle_arch_irq variable。

13. restart

in arch/arm/kernel/setup.c, setup_arch()

```
	if (mdesc->restart)
		arm_pm_restart = mdesc->restart;
```

而arm_pm_restart定义在arch/arm/kernel/process.c中

```
void (*arm_pm_restart)(enum reboot_mode reboot_mode, const char *cmd);
```

```
/*
```

```
 * Restart requires that the secondary CPUs stop performing any activity
 * while the primary CPU resets the system. Systems with a single CPU can
 * use soft_restart() as their machine descriptor's .restart hook, since that
 * will cause the only available CPU to reset. Systems with multiple CPUs must
 * provide a HW restart implementation, to ensure that all CPUs reset at once.
 * This is required so that any code running after reset on the primary CPU
 * doesn't have to co-ordinate with other CPUs to ensure they aren't still
 * executing pre-reset code, and using RAM that the primary CPU's code wishes
 * to use. Implementing such co-ordination would be essentially impossible.
 */
void machine_restart(char *cmd)
{
	local_irq_disable();
	smp_send_stop();

	if (arm_pm_restart)
		arm_pm_restart(reboot_mode, cmd);
	else
		do_kernel_restart(cmd);

	/* Give a grace period for failure to restart of 1s */
	mdelay(1000);

	/* Whoops - the platform was unable to reboot. Tell the user! */
	printk("Reboot failed -- System halted\n");
```

```
    local_irq_disable();
    while (1);
}
```

总结如下：

除了handle_irq和restart是事件trigger外，其他callba ck function在kernel启动时是由不同的调用次序的。

handle_irq --- 由hardware interrupt trigger

restart --- 由machine_restart() trigger

其他callback的invocation从最早到最晚依次为 (fixup在这里忽略)

start_kernel()

setup_arch()

1. dt_fixup()
2. init_meminfo()
3. reserve()
4. map_io()
5. init_early()

.....

6. init_irq()

7. init_time()

.....

rest_init()

kernel_init()

kernel_init_freeable()

do_basic_setup()

do_initcalls()

8. init_machine() (_define_initcall(fn, 3))

9. init_late() (__define_initcall(fn, 7))