MACHINE_START 与 MACHINE_END

在移植 Linux 时，有个结构体需要填写，它以 MACHINE_START 开始并以 MACHINE_END 结束，如下 mini2440 开发板的移植为示例

[cpp] view plaincopy

```cpp
MACHINE_START(MINI2440, "MINI2440")
    .phys_io    = S3C2410_PA_UART,
    .io_pg_offst    = (((u32)S3C24XX_VA_UART) >> 18) & 0xfffc,
    .boot_params    = S3C2410_SDRAM_PA + 0x100,
    .map_io     = mini2440_map_io,
    .init_machine   = mini2440_init,
    .init_irq   = s3c24xx_init_irq,
    .timer      = &s3c24xx_timer,
MACHINE_END
```

其中 MACHINE_START、MACHINE_END 都是定义的宏，代码如下

[cpp] view plaincopy

```cpp
/*
 * Set of macros to define architecture features.  This is built into
 * a table by the linker.
 */
#define MACHINE_START(_type,_name)          \
static const struct machine_desc __mach_desc_##_type    \
 __used                          \
 __attribute__((__section__(".arch.info.init"))) = {    \
    .nr     = MACH_TYPE_##_type,        \
    .name       = _name,

#define MACHINE_END             \
};
```

由上代码可知这两个宏一起定义了一个类型为 struct machine_desc 的变量，结构体定义如下

[cpp] view plaincopy

```cpp
struct machine_desc {
    /*
     * Note! The first four elements are used
     * by assembler code in head.S, head-common.S
     */
    unsigned int        nr;     /* architecture number  */
    unsigned int        phys_io;    /* start of physical io */
    unsigned int        io_pg_offst;    /* byte offset for io
                         * page tabe entry  */

    const char      *name;      /* architecture name    */
    unsigned long       boot_params;    /* tagged list      */

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

    unsigned int        reserve_lp0 :1; /* never has lp0    */
    unsigned int        reserve_lp1 :1; /* never has lp1    */
    unsigned int        reserve_lp2 :1; /* never has lp2    */
    unsigned int        soft_reboot :1; /* soft reboot      */
    void            (*fixup)(struct machine_desc *,
                     struct tag *, char **,
                     struct meminfo *);
    void            (*map_io)(void);/* IO mapping function  */
    void            (*init_irq)(void);
    struct sys_timer    *timer;     /* system tick timer    */
    void            (*init_machine)(void);
};
```

这个类型的变量放在内核代码段.arch.info.init 中，在内核运行初期，被函数 lookup_machine_type（此函数用汇编实现，在汇编文件中）取出，读取流程为

Start_kernel() -> setup_arch() -> setup_machine() -> lookup_machine_type()

在函数 setup_machine()中，利用这个结构体类型的变量初始化一些全局变量，以备内核运行时使用，比如

```
init_arch_irq = mdesc->init_irq;
        system_timer = mdesc->timer;
        init_machine = mdesc->init_machine;
```

这个结构体中，成员 init_machine 保存的是开发板资源注册的初始化代码，init_irq 保存的是中断初始化指针，timer 保存的是一个 struct sys_timer 类型的指针.....如果我们要给自己的开发板定制内核，那么我们必须自己实现以上成员函数，其中函数 init_machine()是我们向内核传递开发板设备信息的重要的常规途径，分析 mini2440 开发板内核移植代码知道，在这个函数中，注册了开发板所用到的所有设备的相关硬件信息！

[cpp] view plaincopy

```cpp
static void __init mini2440_init(void)
{
    struct mini2440_features_t features = { 0 };
    int i;

    printk(KERN_INFO "MINI2440: Option string mini2440=%s\n",
            mini2440_features_str);

    /* Parse the feature string */
    mini2440_parse_features(&features, mini2440_features_str);

    /* turn LCD on */
    s3c_gpio_cfgpin(S3C2410_GPC(0), S3C2410_GPC0_LEND);

    /* Turn the backlight early on */
    WARN_ON(gpio_request(S3C2410_GPG(4), "backlight"));
    gpio_direction_output(S3C2410_GPG(4), 1);
```

```
18.
19.     /* remove pullup on optional PWM backlight -- unused on 3.5 and 7"s */
20.     s3c_gpio_setpull(S3C2410_GPB(1), S3C_GPIO_PULL_UP);
21.     s3c2410_gpio_setpin(S3C2410_GPB(1), 0);
22.     s3c_gpio_cfgpin(S3C2410_GPB(1), S3C2410_GPIO_INPUT);
23.
24.     /* Make sure the D+ pullup pin is output */
25.     WARN_ON(gpio_request(S3C2410_GPC(5), "udc pup"));
26.     gpio_direction_output(S3C2410_GPC(5), 0);
27.
28.     /* mark the key as input, without pullups (there is one on the board) */

29.     for (i = 0; i < ARRAY_SIZE(mini2440_buttons); i++) {
30.         s3c_gpio_setpull(mini2440_buttons[i].gpio, S3C_GPIO_PULL_UP);
31.         s3c_gpio_cfgpin(mini2440_buttons[i].gpio, S3C2410_GPIO_INPUT);
32.     }
33.     if (features.lcd_index != -1) {
34.         int li;
35.
36.         mini2440_fb_info.displays =
37.             &mini2440_lcd_cfg[features.lcd_index];
38.
39.         printk(KERN_INFO "MINI2440: LCD");
40.         for (li = 0; li < ARRAY_SIZE(mini2440_lcd_cfg); li++)
41.             if (li == features.lcd_index)
42.                 printk(" [%d:%dx%d]", li,
43.                     mini2440_lcd_cfg[li].width,
44.                     mini2440_lcd_cfg[li].height);
45.             else
46.                 printk(" %d:%dx%d", li,
47.                     mini2440_lcd_cfg[li].width,
48.                     mini2440_lcd_cfg[li].height);
49.         printk("\n");
50.         s3c24xx_fb_set_platdata(&mini2440_fb_info);
51.     }
52.
53.     s3c24xx_udc_set_platdata(&mini2440_udc_cfg);
54.     s3c24xx_mci_set_platdata(&mini2440_mmc_cfg);
55.     s3c_nand_set_platdata(&mini2440_nand_info);
56.     s3c_i2c0_set_platdata(NULL);
57.
58.     i2c_register_board_info(0, mini2440_i2c_devs,
59.                 ARRAY_SIZE(mini2440_i2c_devs));
60.
```

```
61.     platform_add_devices(mini2440_devices, ARRAY_SIZE(mini2440_devices));
62.
63.     if (features.count) /* the optional features */
64.         platform_add_devices(features.optional, features.count);
65.
66. }
```

那么成员函数 init_machine 什么时候被调用呢？

在函数 setup_machine()中有一条语句 init_machine = mdesc->init_machine;其中 init_machine 为全局函数指针变量，此变量在函数 customize_machine()中被调用,代码如下所示：

[cpp] view plaincopy

```
1.  static int __init customize_machine(void)
2.
3.  {
4.
5.          /* customizes platform devices, or adds new ones */
6.
7.          if (init_machine)
8.
9.                  init_machine();
10.
11.         return 0;
12.
13. }
14.
15. arch_initcall(customize_machine);
```

在 MACHINE_START 与 MACHINE_END 之间还要填写一些参数，参照结构体注释小心填写即可，最好找个例子参考参考。