U-boot 源代码分析
（for AT91SAM9260EK）

作者：张俊岭
EMAIL: sprite_zjl@sina.com
MSN: sprite_zjl@hotmail.com
QQ: 251450387
版本：1.0
日期：2007-11-8
摘要：
本文档基于 AT91SAM9260EK 板，详细分析了 U-boot-1.1.4 的初始化过程、命令处理过程及 linux 系统的引导过程。

1 第一阶段（Stage 1）

第一阶段的启动代码在 cpu\<cpu type>\start.s 中，完成的工作主要有：

- CPU 自身初始化：包括 MMU，Cache，时钟系统，SDRAM 控制器等的初始化
- 重定位：把自己从非易失性存储器搬移到 RAM 中
- 分配堆栈空间，设置堆栈指针
- 清零 BSS 数据段
- 跳转到第二阶段入口函数 start_armboot()

AT91SAM9260EK 的启动代码在 cpu\arm926ejs\start.s 中，精简后的代码如下：
[cpu\arm926ejs\start.s]

```
; ARM 的向量表
.globl _start
_start:
    b     reset
    ldr   pc, _undefined_instruction
    ldr   pc, _software_interrupt
    ldr   pc, _prefetch_abort
    ldr   pc, _data_abort
    ldr   pc, _not_used
    ldr   pc, _irq
    ldr   pc, _fiq

_undefined_instruction:
    .word undefined_instruction
_software_interrupt:
    .word software_interrupt
_prefetch_abort:
```

```
	.word prefetch_abort
_data_abort:
	.word data_abort
_not_used:
	.word not_used
_irq:
	.word irq
_fiq:
	.word fiq


;全局符号定义
_TEXT_BASE:
	.word	TEXT_BASE
.globl _armboot_start
_armboot_start:
	.word _start

/*
 * These are defined in the board-specific linker script.
 */
.globl _bss_start
_bss_start:
	.word __bss_start
.globl _bss_end
_bss_end:
	.word _end

#ifdef	CONFIG_USE_IRQ
/* IRQ stack memory (calculated at run-time) */
.globl	IRQ_STACK_START
IRQ_STACK_START:
	.word	0x0badc0de

/* IRQ stack memory (calculated at run-time) */
.globl FIQ_STACK_START
FIQ_STACK_START:
	.word 0x0badc0de
#endif

;复位入口
reset:
	; CPU 设为 SVC32 模式
	mrs  r0,cpsr
```

```
        bic   r0,r0,#0x1f
        orr   r0,r0,#0xd3
        msr  cpsr,r0

;如果需要，调用 cpu_init_crit 进行 CPU 关键初始化
;在 AT91SAM9260EK 板上没有使用。这部分工作在 Bootstrap 中完成。
#ifndef    CONFIG_SKIP_LOWLEVEL_INIT
        bl     cpu_init_crit
#endif



;如果需要，对 U-boot 进行重定位（从 Flash 搬移到 SDRAM 中）
;在 AT91SAM9260EK 板上没有使用。U-boot 在运行之前已经被 Bootstrap 加载到了 SDRAM
中。
#ifndef CONFIG_SKIP_RELOCATE_UBOOT
relocate:                    /* relocate U-Boot to RAM          */
        adr   r0, _start        /* r0 <- current position of code     */
        ldr   r1, _TEXT_BASE        /* test if we run from flash or RAM */
        cmp          r0, r1                          /* don't reloc during debug               */
        beq          stack_setup
        ldr   r2, _armboot_start
        ldr   r3, _bss_start
        sub   r2, r3, r2        /* r2 <- size of armboot                  */
        add   r2, r0, r2        /* r2 <- source end address               */
copy_loop:
        ldmia       r0!, {r3-r10}         /* copy from source address [r0]       */
        stmia       r1!, {r3-r10}         /* copy to      target address [r1]       */
        cmp r0, r2                  /* until source end addreee [r2]       */
        ble   copy_loop
#endif     /* CONFIG_SKIP_RELOCATE_UBOOT */

;为 irq，fiq，abt 模式分配堆栈
stack_setup:
        ldr   r0, _TEXT_BASE                    ;指向 U-boot 起始点
        sub   r0, r0, #CFG_MALLOC_LEN      ;留出 malloc 内存空间
        sub   r0, r0, #CFG_GBL_DATA_SIZE    ;留出 u-boot 私有数据的空间

;如果使用中断机制，分配 irq，fiq 模式的堆栈
;AT91SAM9260EK 不使用中断
#ifdef CONFIG_USE_IRQ
        sub   r0, r0, #(CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ)
#endif
;分配 abt 模式堆栈空间（12 bytes），设置 svc 模式 SP
sub   sp, r0, #12            /* leave 3 words for abort-stack       */
```

```
;清零 BSS 数据段
clear_bss:
        ldr    r0, _bss_start          /* find start of bss segment              */
        ldr    r1, _bss_end            /* stop here                                  */
        mov        r2, #0x00000000             /* clear                                      */

clbss_l:str        r2, [r0]            /* clear loop...                          */
        add  r0, r0, #4
        cmp r0, r1
        ble    clbss_l

;跳转到第二阶段入口 start_armboot()
        ldr    pc, _start_armboot
_start_armboot:
        .word start_armboot

; 以下是 cpu_init_crit 函数
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
cpu_init_crit:

        ;Flush Cache，TLB
        mov r0, #0
        mcr  p15, 0, r0, c7, c7, 0 /* flush v3/v4 cache */
        mcr  p15, 0, r0, c8, c7, 0 /* flush v4 TLB */
        ;禁止 MMU，打开 I-Cache
        mrc  p15, 0, r0, c1, c0, 0
        bic    r0, r0, #0x00002300       /* clear bits 13, 9:8 (--V- --RS) */
        bic    r0, r0, #0x00000087       /* clear bits 7, 2:0 (B--- -CAM) */
        orr    r0, r0, #0x00000002       /* set bit 2 (A) Align */
        orr    r0, r0, #0x00001000       /* set bit 12 (I) I-Cache */
        mcr  p15, 0, r0, c1, c0, 0

        ;调用 lowlevel_init 完成底层初始化（时钟系统，SDRAM 控制器，片选设置等等）
        mov ip, lr        /* perserve link reg across call */
        bl     lowlevel_init   /* go setup pll,mux,memory */
        mov lr, ip        /* restore link */
        mov pc, lr              /* back to my caller */
#endif
```

2 第二阶段（Stage 2）

第二阶段是 u-boot 的主体，入口点是 lib_arm\board.c 中的 start_armboot()函数，完成的主要工作包括：

- 为 U-boot 内部私有数据分配存储空间，并清零
- 依次调用函数指针数组 init_sequence 中定义的函数进行一系列的初始化
- 如果系统支持 NOR Flash，调用 flash_init ()和 display_flash_config ()初始化并显示检测到的器件信息（**AT91SAM9260EK 不需要**）
- 如果系统支持 LCD 或 VFD,调用 lcd_setmem()或 vfd_setmem()计算帧缓冲(Framebuffer)大小，然后在 BSS 数据段之后为 Framebuffer 分配空间,初始化 gd->fb_base 为 Framebuffer 的起始地址（**AT91SAM9260EK 不需要**）
- 调用 mem_malloc_init()进行存储分配系统（类似于 C 语言中的堆）的初始化和空间分配
- 如果系统支持 NAND Flash，调用 nand_init ()进行初始化
- 如果系统支持 DataFlash，调用 AT91F_DataflashInit()和 dataflash_print_info()进行初始化并显示检测到的器件信息
- 调用 env_relocate ()进行环境变量的重定位，即从 Flash 中搬移到 RAM 中
- 如果系统支持 VFD，调用 drv_vfd_init()进行 VFD 设备初始化（**AT91SAM9260EK 不需要**）
- 从环境变量中读取 IP 地址和 MAC 地址，初始化 gd->bd-> bi_ip_addr 和 gd->bd->bi_enetaddr
- 调用 jumptable_init ()进行跳转表初始化，跳转表在 global_data 中，具体用途尚不清楚
- 调用 console_init_r()进行控制台初始化
- 如果需要，调用 misc_init_r ()进行杂项初始化
- 调用 enable_interrupts ()打开中断
- 如果需要，调用 board_late_init()进行单板后期初始化，对于 AT91SAM9260EK，主要是以太网初始化
- 进入主循环：根据用户的选择启动 linux，或者进入命令循环执行用户输入的命令

源代码：

[lib_arm\board.c]

```
void start_armboot (void)
{
    DECLARE_GLOBAL_DATA_PTR;
/*
  在 include/asm-arm/global_data.h 中定义：
  #define DECLARE_GLOBAL_DATA_PTR     register volatile gd_t *gd asm ("r8")
  声明了一个变量 gd，保存在寄存器 r8 中，类型为 gd_t(即 struct global_data)指针
*/

    ulong size;
```

```
    init_fnc_t **init_fnc_ptr;
    char *s;
#if defined(CONFIG_VFD) || defined(CONFIG_LCD)
    unsigned long addr;
#endif

/* 为 global_data 分配空间，并清零  */
    gd = (gd_t*)(_armboot_start - CFG_MALLOC_LEN - sizeof(gd_t));
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("": : :"memory");
    memset ((void*)gd, 0, sizeof (gd_t));
    gd->bd = (bd_t*)((char*)gd - sizeof(bd_t));
    memset (gd->bd, 0, sizeof (bd_t));

    monitor_flash_len = _bss_start - _armboot_start;

/*
依次调用函数指针数组 init_sequence 中定义的函数，如果中途出错，调用 hung()进入死循环
*/
    for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {
        if ((*init_fnc_ptr)() != 0) {
            hang ();
        }
    }
/* 初始化 NOR Flash， AT91SAM9260EK 不需要 */
#if (CONFIG_COMMANDS & CFG_CMD_FLASH)
    /* configure available FLASH banks */
    size = flash_init ();
    display_flash_config (size);
#endif

/* 为 LCD 或 VFD 分配 Framebuffer，AT91SAM9260EK 不需要 */
#ifdef CONFIG_VFD
#   ifndef PAGE_SIZE
#     define PAGE_SIZE 4096
#   endif
    /*
     * reserve memory for VFD display (always full pages)
     */
    /* bss_end is defined in the board-specific linker script */
    addr = (_bss_end + (PAGE_SIZE - 1)) & ~(PAGE_SIZE - 1);
    size = vfd_setmem (addr);
    gd->fb_base = addr;
```

```
#endif /* CONFIG_VFD */

#ifdef CONFIG_LCD
#   ifndef PAGE_SIZE
#     define PAGE_SIZE 4096
#   endif

#if !defined(CONFIG_MACH_AT91SAM9261EK) && !defined(CONFIG_MACH_NADIA2VB)
    /*
     * reserve memory for LCD display (always full pages)
     */
    /* bss_end is defined in the board-specific linker script */
    addr = (_bss_end + (PAGE_SIZE - 1)) & ~(PAGE_SIZE - 1);
    size = lcd_setmem (addr);
    gd->fb_base = addr;
#endif
#endif /* CONFIG_LCD */

/* 初始化存储分配系统（简化的堆） */
    mem_malloc_init (_armboot_start - CFG_MALLOC_LEN);

/* 初始化 NAND Flash  */
#if (CONFIG_COMMANDS & CFG_CMD_NAND)
    nand_init();        /* go init the NAND */
#endif

/* 初始化 DataFlash  */
#ifdef CONFIG_HAS_DATAFLASH
    AT91F_DataflashInit();
    dataflash_print_info();
#endif

/* 环境变量重定位  */
    /* initialize environment */
    env_relocate ();

/* VFD 设备初始化，AT91SAM9260EK 不需要  */
#ifdef CONFIG_VFD
    /* must do this after the framebuffer is allocated */
    drv_vfd_init();
#endif /* CONFIG_VFD */

    puts ("IP Address \n");
/* 从环境变量"ipaddr"中读取 IP 地址，初始化 gd->bd->bi_ip_addr */
```

```
    gd->bd->bi_ip_addr = getenv_IPaddr ("ipaddr");

/* 从环境变量"ethaddr"中读取 MAC 地址，初始化 gd->bd->bi_enetaddr */
    {
        int i;
        ulong reg;
        char *s, *e;
        uchar tmp[64];

        i = getenv_r ("ethaddr", tmp, sizeof (tmp));
        s = (i > 0) ? tmp : NULL;

        for (reg = 0; reg < 6; ++reg) {
            gd->bd->bi_enetaddr[reg] = s ? simple_strtoul (s, &e, 16) : 0;
            if (s)
                s = (*e) ? e + 1 : e;
        }
    }

/* 初始化设备 */
 devices_init ();   /* get the devices list going. */

#ifdef CONFIG_CMC_PU2
    load_sernum_ethaddr ();
#endif /* CONFIG_CMC_PU2 */

/* 初始化跳转表 */
    jumptable_init ();

/* 初始化控制台 */
    console_init_r ();  /* fully init console as a device */

/* 杂项初始化 */
#if defined(CONFIG_MISC_INIT_R)
    /* miscellaneous platform dependent initialisations */
    misc_init_r ();
#endif

/* 开中断 */
    /* enable exceptions */
    enable_interrupts ();

    /* Perform network card initialisation if necessary */
/* CS8900 网卡，SMC91111 初始化，AT91SAM9260EK 不需要 */
```

```
#ifdef CONFIG_DRIVER_CS8900
    cs8900_get_enetaddr (gd->bd->bi_enetaddr);
#endif
#if defined(CONFIG_DRIVER_SMC91111) || defined (CONFIG_DRIVER_LAN91C96)
    if (getenv ("ethaddr")) {
        smc_set_mac_addr(gd->bd->bi_enetaddr);
    }
#endif /* CONFIG_DRIVER_SMC91111 || CONFIG_DRIVER_LAN91C96 */

/* 读取环境变量"loadaddr"到 loadaddr 变量 */
    /* Initialize from environment */
    if ((s = getenv ("loadaddr")) != NULL) {
        load_addr = simple_strtoul (s, NULL, 16);
    }
/* 读取环境变量"bootfile"到 BootFile 变量 */
#if (CONFIG_COMMANDS & CFG_CMD_NET)
    if ((s = getenv ("bootfile")) != NULL) {
        copy_filename (BootFile, s, sizeof (BootFile));
    }
#endif  /* CFG_CMD_NET */

/* 单板后期初始化 */
#ifdef BOARD_LATE_INIT
    board_late_init ();
#endif
#if (CONFIG_COMMANDS & CFG_CMD_NET)
#if defined(CONFIG_NET_MULTI)
    puts ("Net:   ");
#endif
    eth_initialize(gd->bd);
#endif

/* 主循环  */
    /* main_loop() can return to retry autoboot, if so just run it again. */
    for (;;) {
        main_loop ();
    }
    /* NOTREACHED - no way out of command loop except booting */
}
```

3 U-boot 的初始化

3．1　私有数据 global_data

global_data 在 include\asm-arm\global_data.h 中定义:

[include\asm-arm\global_data.h]

```
typedef    struct      global_data {
      bd_t        *bd;              /* bd_info 结构，包含更多的信息 */
      unsigned long  flags;
      unsigned long  baudrate;     /* 波特率 */
      unsigned long  have_console; /* 有无控制台 */
      unsigned long  reloc_off;  /* 重定位的偏移量 */
      unsigned long  env_addr;  /* 环境变量块的地址 */
      unsigned long  env_valid;      /* 环境变量是否有效 */
      unsigned long  fb_base;  /* Frame buffer 地址 */
#ifdef CONFIG_VFD
      unsigned char  vfd_type;       /* VFD 类型 */
#endif
      void        **jt;       /* 跳转表 */
} gd_t;
```

bd_info 在 include\asm-arm\u-boot.h 中定义:

[include\asm-arm\u-boot.h]

```
typedef struct bd_info {
      int               bi_baudrate;   /* 串行口波特率 */
      unsigned long  bi_ip_addr;     /* IP 地址 */
      unsigned char  bi_enetaddr[6]; /* MAC 地址 */
      struct environment_s                *bi_env;    /* 环境变量块地址    */
      ulong                  bi_arch_number;    /* 机器类型     */
      ulong                  bi_boot_params;    /* 传递给 linux 的参数块地址 */
      struct                       /* RAM configuration */
      {
      ulong start;
      ulong size;
      }bi_dram[CONFIG_NR_DRAM_BANKS];   /* DRAM 区间列表：起始地址和大小 */
} bd_t;
```

3．2　初始化序列 init_sequence

Init_sequence 是一个函数指针数组，数组中每一个元素都指向一个初始化函数。

[lib_arm/board.c]

```
typedef int (init_fnc_t) (void);
init_fnc_t *init_sequence[] = {
      cpu_init,         /* basic cpu dependent setup */
      board_init,             /* basic board dependent setup */
```

```
        interrupt_init,         /* set up exceptions */
        env_init,       /* initialize environment */
        init_baudrate,          /* initialze baudrate settings */
        serial_init,            /* serial communications setup */
        console_init_f,         /* stage 1 init of console */
        display_banner,             /* say that we are here */
        dram_init,              /* configure available RAM banks */
        display_dram_config,
#if defined(CONFIG_VCMA9) || defined (CONFIG_CMC_PU2)
        checkboard,
#endif
        NULL,
};
```

对于 AT91SAM9260EK，这些函数的位置和完成的功能如表 1 所示：

表 1　init_sequence 中初始化函数说明

函数名称	位置（所在的文件）	功能概述
cpu_init	cpu/arm926ejs/cpu.c	堆栈初始化
board_init	board/at91sam9260ek/at91sam9260ek.c	复位 PHY 芯片；设置 GPIO 口；设置机器类型和启动参数块地址到 global_data 中相关数据成分中
interrupt_init	cpu/arm926ejs/at91sam9260/interrupts.c	初始化 PIT（可编程间隔定时器）
env_init	common/env_dataflash.c	初始化 Dataflash；判断 DataFlash 中环境变量快的有效性（计算比较 CRC），设置 gd->env_valid；设置环境变量快的起始地址 gd->env_addr
init_baudrate	lib_arm/board.c	从环境变量中读取波特率，初始化 gd->bd->bi_baudrate
serial_init	cpu/arm926ejs/at91sam9260/serial.c	初始化串行口
console_init_f	common/console.c	控制台初始化第一阶段：初始化 gd->have_console 和 gd->flags
display_banner	lib_arm/board.c	输出版本信息和代码/数据段/堆栈信息到控制台
dram_init	board/at91sam9260ek/at91sam9260ek.c	初始化 SDRAM 区间信息：gd->bd->bi_dram
display_dram_config	lib_arm/board.c	输出 SDRAM 区间信息信息到控制台
checkboard	-	AT91SAM9260EK 未使用

下面重点分析 cpu_init,board_init,interrupt_init,env_init,serial_init 等几个函数,其它几个比较简单，这里不再分析。

(1) cpu_init

这个函数的功能是设置 irq 和 fiq 模式的堆栈起始点。 AT91SAM9260EK 没有使用 U-boot

的中断机制，所以这个函数实际上什么也没做。

[cpu/arm926ejs/cpu.c]

```
int cpu_init (void)
{
      /*
       * setup up stacks if necessary
       */
#ifdef CONFIG_USE_IRQ
DECLARE_GLOBAL_DATA_PTR;

IRQ_STACK_START = _armboot_start - CFG_MALLOC_LEN - CFG_GBL_DATA_SIZE - 4;
FIQ_STACK_START = IRQ_STACK_START - CONFIG_STACKSIZE_IRQ;
#endif
return 0;
}
```

(2) board_init

[board/at91sam9260ek/at91sam9260ek.c]

```
int board_init (void)
{
  DECLARE_GLOBAL_DATA_PTR;
  /*   复位 PHY 芯片，复位脉冲宽度 500ms   */
  AT91C_BASE_RSTC->RSTC_RMR = (AT91C_RSTC_KEY & ((unsigned int)0xA5<<24)) |
                              (AT91C_RSTC_ERSTL & (0x0D << 8));
  AT91C_BASE_RSTC->RSTC_RCR = (AT91C_RSTC_KEY & ((unsigned int)0xA5<<24)) |
                               AT91C_RSTC_EXTRST;
 /*   等待复位完成  */
  /* Wait for end hardware reset */
  while (!(AT91C_BASE_RSTC->RSTC_RSR & AT91C_RSTC_NRSTL));

  /*   控制台初始化第一阶段   */
  console_init_f ();

 /*   打开系统控制模块和 PIOC 模块的时钟  */
  //Enable clocks for SMC and PIOC
  AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_SYS;
  AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_PIOC;

  //bright all led     su.2006.05.29
   /*   设置 GPIO 口，点亮相关的 LED */
AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,AT91C_PIO_PA15|AT91C_PIO_PA16|AT91C_P
```

```
IO_PA30|AT91C_PIO_PA31);
AT91F_PIO_SetOutput(AT91C_BASE_PIOA,AT91C_PIO_PA23);
 /*  如果配置了 LCD，设置 Framebuffer 起始地址 */
#ifdef CONFIG_LCD
#ifndef CONFIG_LCD_EXT_SDRAM
   gd->fb_base = (unsigned long) AT91C_IRAM;
#else
   gd->fb_base = (unsigned long) AT91C_EXT_SDRAM;
#endif
#endif

 /*  设置机器类型，MACH_TYPE_AT91SAM9260EK 在 include/asm-arm/mach-types.h
     中定义为 848
 */
   gd->bd->bi_arch_number = MACH_TYPE_AT91SAM9260EK;

   /* 设置引导参数块地址 0x20000100      */
   gd->bd->bi_boot_params = PHYS_SDRAM + 0x100;

   return 0;
}
```

(3) interrupt_init

[cpu/arm926ejs/at91sam9260/interrupts.c]

```
int interrupt_init (void)
{
   p_pitc = AT91C_BASE_PITC;

   /* 打开系统控制模块的时钟 */
    *AT91C_PMC_PCER = 1 << AT91C_ID_SYS;

   /* 打开 PIT(周期间隔计时器)   */
   p_pitc->PITC_PIMR = AT91C_PITC_PITEN;

   /* 设置 PIT 的计数初始值   */
   p_pitc->PITC_PIMR |= TIMER_LOAD_VAL;

   /* 复位 PIT   */
   reset_timer_masked();

   return 0;
}
```

(4) env_init

[common/env_dataflash.c]

```
int env_init(void)
{
   DECLARE_GLOBAL_DATA_PTR;
   ulong crc, len, new;
   unsigned off;
   uchar buf[64];
   int DataflashExists;

   if (gd->env_valid == 0){
   /*   DataFlash 初始化，检测  DataFlash 是否存在  */
    if((DataflashExists = AT91F_DataflashInit())){      /* prepare for DATAFLASH read/write */
        //printf("Dataflash is not plugged or not supported\n");
        //return 1;
        //}

         /*   从 DataFlash 读入环境变量块的 CRC 值    */
        read_dataflash (CFG_ENV_ADDR+offsetof(env_t,crc),sizeof(ulong),&crc);
        /*   分段从 DataFlash 读入环境变量块，每次读取 64 字节，计算 CRC    */
        new = 0;
        len = ENV_SIZE;
        off = offsetof(env_t,data);
        while (len > 0) {
      int n = (len > sizeof(buf)) ? sizeof(buf) : len;
      read_dataflash (CFG_ENV_ADDR+off,n , buf);
      new = crc32 (new, buf, n);
      len -= n;
      off += n;
         }
          /*
            如果 CRC 正确（计算结果和读取的值相等），设置 gd->env_valid=1,gd->env_addr
            指向环境变量数据区起始地址（注意此时环境变量尚未读入 RAM）
          */
        if (crc == new) {
      gd->env_addr    = offsetof(env_t,data);
      gd->env_valid = 1;
      }
      /*  否则,设置 gd->env_valid=0,gd->env_addr 指向缺省环境变量          */
      else {
      gd->env_addr    = (ulong)&default_environment[0];
      gd->env_valid = 0;
```

```
        }
      }
   }

   return 0;
}
```

缺省环境变量 default_environment 在 common/env_common.c 中被定义为：

```
uchar default_environment[] = {
#ifdef      CONFIG_BOOTARGS
      "bootargs="     CONFIG_BOOTARGS               "\0"
#endif
#ifdef      CONFIG_BOOTCOMMAND
      "bootcmd="      CONFIG_BOOTCOMMAND            "\0"
#endif
#ifdef      CONFIG_RAMBOOTCOMMAND
      "ramboot="      CONFIG_RAMBOOTCOMMAND         "\0"
#endif
#ifdef      CONFIG_NFSBOOTCOMMAND
      "nfsboot="      CONFIG_NFSBOOTCOMMAND         "\0"
#endif
#if defined(CONFIG_BOOTDELAY) && (CONFIG_BOOTDELAY >= 0)
      "bootdelay="    MK_STR(CONFIG_BOOTDELAY)      "\0"
#endif
#if defined(CONFIG_BAUDRATE) && (CONFIG_BAUDRATE >= 0)
      "baudrate="     MK_STR(CONFIG_BAUDRATE)       "\0"
#endif
#ifdef      CONFIG_LOADS_ECHO
      "loads_echo="   MK_STR(CONFIG_LOADS_ECHO)    "\0"
#endif
#ifdef      CONFIG_ETHADDR
      "ethaddr="      MK_STR(CONFIG_ETHADDR)        "\0"
#endif
#ifdef      CONFIG_ETH1ADDR
      "eth1addr="     MK_STR(CONFIG_ETH1ADDR)       "\0"
#endif
#ifdef      CONFIG_ETH2ADDR
      "eth2addr="     MK_STR(CONFIG_ETH2ADDR)       "\0"
#endif
#ifdef      CONFIG_ETH3ADDR
      "eth3addr="     MK_STR(CONFIG_ETH3ADDR)       "\0"
#endif
#ifdef      CONFIG_IPADDR
      "ipaddr="MK_STR(CONFIG_IPADDR)                "\0"
```

```
#endif
#ifdef      CONFIG_SERVERIP
      "serverip="     MK_STR(CONFIG_SERVERIP)         "\0"
#endif
#ifdef      CFG_AUTOLOAD
      "autoload="     CFG_AUTOLOAD              "\0"
#endif
#ifdef      CONFIG_PREBOOT
      "preboot="      CONFIG_PREBOOT                  "\0"
#endif
#ifdef      CONFIG_ROOTPATH
      "rootpath="     MK_STR(CONFIG_ROOTPATH)       "\0"
#endif
#ifdef      CONFIG_GATEWAYIP
      "gatewayip="  MK_STR(CONFIG_GATEWAYIP)      "\0"
#endif
#ifdef      CONFIG_NETMASK
      "netmask="      MK_STR(CONFIG_NETMASK)        "\0"
#endif
#ifdef      CONFIG_HOSTNAME
      "hostname="   MK_STR(CONFIG_HOSTNAME)           "\0"
#endif
#ifdef      CONFIG_BOOTFILE
      "bootfile="     MK_STR(CONFIG_BOOTFILE)       "\0"
#endif
#ifdef      CONFIG_LOADADDR
      "loadaddr="     MK_STR(CONFIG_LOADADDR)           "\0"
#endif
#ifdef    CONFIG_CLOCKS_IN_MHZ
      "clocks_in_mhz=1\0"
#endif
#if defined(CONFIG_PCI_BOOTDELAY) && (CONFIG_PCI_BOOTDELAY > 0)
      "pcidelay="     MK_STR(CONFIG_PCI_BOOTDELAY)    "\0"
#endif
#ifdef    CONFIG_EXTRA_ENV_SETTINGS
      CONFIG_EXTRA_ENV_SETTINGS
#endif
      "\0"
};
```

可以看出，这是所有环境变量的缺省值，可以由用户定义。重要的环境变量如表 2 所示：

表 2　环境变量缺省值

环境变量名称	环境变量值
bootargs	"mem=64M console=ttyS0,115200 initrd=0x21100000,17000000 root=/dev/ram0 rw"
bootcmd	"nand read 20400000 0 200000;nand read 21100000 200000 400000;bootm 20400000"
bootdelay	"3"
baudrate	"115200"
ethaddr	"22.34.56.78.99.aa"
ipaddr	"192.168.12.138"
serverip	"192.168.12.66"
autoload	"tftp 20400000 ulmage9260;tftp 21100000 ramdisk9260.gz;bootm 20400000"

(5)　serial_init

[cpu/arm926ejs/at91sam9260/serial.c]

```
int serial_init (void)
{
    /* 如果使用调试串口，配置调试串口的管脚 */
  #ifdef CONFIG_DBGU
  *AT91C_PIOB_PDR = AT91C_PB15_DTXD | AT91C_PB14_DRXD;  /* PA 10 & 9 */
  *AT91C_PMC_PCER = 1 << AT91C_ID_SYS;      /* enable clock */
#endif
    /* 如果使用串口 0，配置串口 0 的管脚 */
#ifdef CONFIG_USART0
  *AT91C_PIOA_PDR = AT91C_PC8_TXD0 | AT91C_PC9_RXD0;
  *AT91C_PMC_PCER |= 1 << AT91C_ID_US0;      /* enable clock */
#endif
    /* 如果使用串口 1，配置串口 1 的管脚 */
#ifdef CONFIG_USART1
  *AT91C_PIOB_PDR = AT91C_PC12_TXD1 | AT91C_PC13_RXD1;
  *AT91C_PMC_PCER |= 1 << AT91C_ID_US1;      /* enable clock */
#endif
    /* 如果使用串口 2，配置串口 2 的管脚 */
#ifdef CONFIG_USART2
  *AT91C_PIOB_PDR = AT91C_PC14_TXD2 | AT91C_PC15_RXD2;
  *AT91C_PMC_PCER |= 1 << AT91C_ID_US2;      /* enable clock */
#endif

  /* 设置串口波特率 */
  serial_setbrg ();
```

```
    /* 串口收发复位，打开收发允许 */
    us->US_CR = AT91C_US_RSTRX | AT91C_US_RSTTX;
    us->US_CR = AT91C_US_RXEN | AT91C_US_TXEN;
    /* 设置串口模式 8-N-1 */
    us->US_MR =
        (AT91C_US_CLKS_CLOCK | AT91C_US_CHRL_8_BITS |
         AT91C_US_PAR_NONE | AT91C_US_NBSTOP_1_BIT);
    /* 屏蔽串口所有中断 */
    us->US_IMR = ~0ul;
    return (0);
}

void serial_setbrg (void)
{
    DECLARE_GLOBAL_DATA_PTR;
    int baudrate;
     /* baudrate = gd->baudrate */
    if ((baudrate = gd->baudrate) <= 0)
        baudrate = CONFIG_BAUDRATE;
    if (baudrate == 0 || baudrate == CONFIG_BAUDRATE)
        //us->US_BRGR = CFG_AT91C_BRGR_DIVISOR;  /* hardcode so no __divsi3 */
    /* 设置串口波特率为 baudrate   */
        us->US_BRGR = AT91C_MASTER_CLOCK/(baudrate * 16);
        //else
    /* MASTER_CLOCK/(16 * baudrate) */
    //    us->US_BRGR = AT91C_MASTER_CLOCK/(baudrate * 16);
}
```

3.3 NAND Flash 初始化

nand_init()完成 NAND Flash 的初始化。这个函数在 board/at91sam9260ek/at91sam9260ek.c 中。

[board/at91sam9260ek/at91sam9260ek.c]

```
void nand_init (void)
{
    /* 分配 CS3 给 NAND Flash 控制器   */
    *AT91C_CCFG_EBICSA |= AT91C_MATRIX_CS3A_SM;
    /* 设置 CS3 时序（建立时间，脉冲宽度，操作周期）和模式   */
    //Configure SMC CS3
    *AT91C_SMC_SETUP3 = (AT91C_SM_NWE_SETUP | AT91C_SM_NCS_WR_SETUP |
                         AT91C_SM_NRD_SETUP | AT91C_SM_NCS_RD_SETUP);

    *AT91C_SMC_PULSE3  =  (AT91C_SM_NWE_PULSE  |  AT91C_SM_NCS_WR_PULSE  |
AT91C_SM_NRD_PULSE | AT91C_SM_NCS_RD_PULSE);
```

```
  *AT91C_SMC_CYCLE3 = (AT91C_SM_NWE_CYCLE | AT91C_SM_NRD_CYCLE);

  *AT91C_SMC_CTRL3  =  (AT91C_SMC_READMODE  |  AT91C_SMC_WRITEMODE  |
AT91C_SMC_NWAITM_NWAIT_DISABLE |
#ifdef     CFG_16BITS_NANDFLASH
                        AT91C_SMC_DBW_WIDTH_SIXTEEN_BITS |
#endif
#ifdef     CFG_8BITS_NANDFLASH
                        AT91C_SMC_DBW_WIDTH_EIGTH_BITS |
#endif
                        AT91C_SM_TDF);

  /* 打开 GPIO PortA 的时钟 */
  AT91F_PIOA_CfgPMC();

  /* 设置 PA7 为输入 & 允许上拉电阻（PA7 为 RD/BY 信号） */
  AT91F_PIO_CfgInput(AT91C_BASE_PIOA, AT91C_PIO_PA7);
  AT91F_PIO_CfgPullup(AT91C_BASE_PIOA, AT91C_PIO_PA7);

   /* 设置 PA6 为输出（PA6 为 CE 信号） */
  AT91F_PIO_CfgOutput(AT91C_BASE_PIOA,AT91C_PIO_PA6);

  /* 设置 PA9 为输出（PA9 为 WP 信号） & 高电平 */
  AT91F_PIO_CfgOutput(AT91C_BASE_PIOA,AT91C_PIO_PA9);
  *AT91C_PIOA_SODR = AT91C_PIO_PA9;

  /* 检测 NAND Flash 是否存在 */
  if(nand_probe(AT91_SMARTMEDIA_BASE))
     printf ("\nNandFlash address : 0x%x\n",AT91_SMARTMEDIA_BASE);
  else
     printf ("\nNo NandFlash detected\n");
}
```

首先初始化 NAND Flash 接口，包括分配片选，设置片选的时序和模式，配置一些相关的 IO 口，然后调用 nand_probe()检测 NAND Flash。nand_probe()函数在 common/cmd_nand.c 中定义(这个文件实现了所有和 NAND Flash 相关的功能)，调用 NanD_ScanChips()函数搜索系统中的 NAND Flash 芯片（NanD_ScanChips()又调用 NanD_IdentChip()检测芯片，实际上就是向 NAND FLASH 芯片发复位和读 ID 命令，根据返回值判断芯片的型号和容量），具体实现细节不再描述。

3.4 DataFlash 初始化

AT91F_DataflashInit()完成 DataFlash 的初始化。这个函数在 drivers/dataflash.c 中。首先调用

AT91F_SpiInit ()初始化 SPI 接口，然后调用 AT91F_DataflashProbe ()扫描所有的 SPI 片选，检测 DataFlash 是否存在，实现原理和 NAND Flash 类似，都是向芯片发送查询 ID 命令，根据返回值判断芯片的类型和容量。

AT91F_SpiInit ()函数的定义在 cpu/arm926ejs/at91sam9260/spi.c 中：

[cpu/arm926ejs/at91sam9260/spi.c]

```
void AT91F_SpiInit(void) {

   volatile unsigned int dummy;

   /* 配置 SPI0 的管脚 */
   AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA,
                       (AT91C_PA0_SPI0_MISO | AT91C_PA1_SPI0_MOSI |
                        AT91C_PA2_SPI0_SPCK | AT91C_PA3_SPI0_NPCS0 |
                        AT91C_PC11_SPI0_NPCS1 | AT91C_PC16_SPI0_NPCS2 |
                        AT91C_PC17_SPI0_NPCS3),
                       0);


    /* 打开 SPI0 的时钟 */
   AT91C_BASE_PMC->PMC_PCER = 1 << AT91C_ID_SPI0;

    /* 复位 SPI0 */
   AT91C_BASE_SPI0->SPI_CR = AT91C_SPI_SWRST;

   /* 设置 SPI0 为 Master 模式，所有片选不激活 */
   /* Configure SPI in Master Mode with No CS selected !!! */
   AT91C_BASE_SPI0->SPI_MR = AT91C_SPI_MSTR | AT91C_SPI_MODFDIS |
AT91C_SPI_PCS;

   /* 设置 SPI0 的 CS0，CS3 的时序和模式 */
   AT91C_BASE_SPI0->SPI_CSR[0] = (AT91C_SPI_CPOL | (AT91C_SPI_DLYBS &
DATAFLASH_TCSS) |
                        (AT91C_SPI_DLYBCT & DATAFLASH_TCHS) |
                        (AT91C_MASTER_CLOCK / AT91C_SPI_CS0_CLK) << 8);

   AT91C_BASE_SPI0->SPI_CSR[3] = (AT91C_SPI_CPOL | (AT91C_SPI_DLYBS &
DATAFLASH_TCSS) |
                        (AT91C_SPI_DLYBCT & DATAFLASH_TCHS) |
                        (AT91C_MASTER_CLOCK / AT91C_SPI_CS3_CLK) << 8);

   /* 允许 SPI0 */
   AT91C_BASE_SPI0->SPI_CR = AT91C_SPI_SPIEN;
   while(!(AT91C_BASE_SPI0->SPI_SR & AT91C_SPI_SPIENS));
```

```
    // Add tempo to get SPI in a safe state.
    // Should be removed for new silicon (Rev B)
    udelay(500000);
    dummy = AT91C_BASE_SPI0->SPI_SR;
    dummy = AT91C_BASE_SPI0->SPI_RDR;
}
```

AT91F_DataflashProbe ()函数的定义在 board/at91sam9260ek/at45.c 中：

[board/at91sam9260ek/at45.c]

```
int AT91F_DataflashProbe(int cs, AT91PS_DataflashDesc pDesc)
{
      /* SPI0 片选允许 */
      AT91F_SpiEnable(cs);
      /* 获取 DataFlash 状态 */
      AT91F_DataFlashGetStatus(pDesc);
      /* 根据返回值判断是否检测到 */
      return((pDesc->command[1] == 0xFF)? 0: pDesc->command[1] & 0x3C);
}

AT91S_DataFlashStatus AT91F_DataFlashGetStatus(AT91PS_DataflashDesc pDesc)
{
      AT91S_DataFlashStatus status;

      /* if a transfert is in progress ==> return 0 */
      if( (pDesc->state) != IDLE)
            return DATAFLASH_BUSY;

      /* first send the read status command (D7H) */
      pDesc->command[0] = DB_STATUS;
      pDesc->command[1] = 0;

      pDesc->DataFlash_state    = GET_STATUS;
      pDesc->tx_data_size      = 0 ; /* Transmit the command and receive response */
      pDesc->tx_cmd_pt              = pDesc->command ;
      pDesc->rx_cmd_pt              = pDesc->command ;
      pDesc->rx_cmd_size            = 2 ;
      pDesc->tx_cmd_size            = 2 ;

       /* 向 SPI 写命令，等待返回 */
      status = AT91F_SpiWrite (pDesc);
      pDesc->DataFlash_state = *( (unsigned char *) (pDesc->rx_cmd_pt) +1);

      return status;
```

```
}
```

3.5 环境变量重定位

common/env_common.c 中的 env_relocate()函数实现环境变量的重定位：

[common/env_common.c]

```
void env_relocate (void)
{
	DECLARE_GLOBAL_DATA_PTR;

	DEBUGF ("%s[%d] offset = 0x%lx\n", __FUNCTION__,__LINE__,
		gd->reloc_off);

#ifdef CONFIG_AMIGAONEG3SE
	enable_nvram();
#endif

	/* 如果环境变量嵌入在代码中，环境变量指针 env_ptr 增加一个重定位偏移 */
#ifdef ENV_IS_EMBEDDED
	/*
	 * The environment buffer is embedded with the text segment,
	 * just relocate the environment pointer
	 */
	env_ptr = (env_t *)((ulong)env_ptr + gd->reloc_off);
	DEBUGF ("%s[%d] embedded ENV at %p\n", __FUNCTION__,__LINE__,env_ptr);
#else
 /* 否则，给环境变量指针 env_ptr 分配存储空间  */
	/*
	 * We must allocate a buffer for the environment
	 */
	env_ptr = (env_t *)malloc (CFG_ENV_SIZE);
	DEBUGF ("%s[%d] malloced ENV at %p\n", __FUNCTION__,__LINE__,env_ptr);
#endif

	/*
	 * After relocation to RAM, we can always use the "memory" functions
	 */
	env_get_char = env_get_char_memory;

/*
 如果环境变量无效（Flash 中存储的环境变量块 CRC 有错），显示“环境变量错误”信息，
 拷贝缺省环境变量
*/
```

```
    if (gd->env_valid == 0) {
#if defined(CONFIG_GTH)   || defined(CFG_ENV_IS_NOWHERE) /*      Environment      not
changable */
            puts ("Using default environment\n\n");
#else
            puts ("*** Warning - bad CRC, using default environment\n\n");
            SHOW_BOOT_PROGRESS (-1);
#endif

            if (sizeof(default_environment) > ENV_SIZE)
            {
                puts ("*** Error - default environment is too large\n\n");
                return;
            }
            memset (env_ptr, 0, sizeof(env_t));
            memcpy (env_ptr->data,
                default_environment,
                sizeof(default_environment));
#ifdef CFG_REDUNDAND_ENVIRONMENT
            env_ptr->flags = 0xFF;
#endif
            env_crc_update ();
            gd->env_valid = 1;
        }
        else {
/*  否则，env_relocate_spec()  调用
    read_dataflash(CFG_ENV_ADDR,CFG_ENV_SIZE,(uchar *)env_ptr)从 Flash 加载环境
    变量
*/
            env_relocate_spec ();
        }
        gd->env_addr = (ulong)&(env_ptr->data);

#ifdef CONFIG_AMIGAONEG3SE
        disable_nvram();
#endif
}
```

3.6　初始化设备

U-boot 中设备的类型是 device_t,，在 include/devices.h 中定义：

```
typedef struct {
    int    flags;                /* Device flags: input/output/system     */
```

```
    int    ext;                 /* Supported extensions                */
    char name[16];              /* Device name                         */

/* GENERAL functions */

    int (*start) (void);        /* To start the device                 */
    int (*stop) (void);         /* To stop the device                  */

/* OUTPUT functions */

    void (*putc) (const char c);    /* To put a char                   */
    void (*puts) (const char *s);   /* To put a string (accelerator)       */

/* INPUT functions */

    int (*tstc) (void);         /* To test if a char is ready...   */
    int (*getc) (void);         /* To get that char                */

/* Other functions */

    void *priv;                 /* Private extensions                  */
} device_t;
```

可以看出，device_t 的主体是一系列操作设备的函数指针，另外还包含了设备名称，标记和私有数据等等。

common/devices.c 中的 devices_init 函数实现设备的初始化：

[common/devices.c]

```
int devices_init (void)
{
#ifndef CONFIG_ARM        /* already relocated for current ARM implementation */
    DECLARE_GLOBAL_DATA_PTR;

    ulong relocation_offset = gd->reloc_off;
    int i;
        for (i = 0; i < (sizeof (stdio_names) / sizeof (char *)); ++i) {
        stdio_names[i] = (char *) (((ulong) stdio_names[i]) +
                                relocation_offset);
    }
#endif

    /*  创建设备链表 devlist */
    devlist = ListCreate (sizeof (device_t));
```

```
    if (devlist == NULL) {
        eputs ("Cannot initialize the list of devices!\n");
        return -1;
    }
    /* 根据需要初始化各个设备，注册到 devlist  */
#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)
    i2c_init (CFG_I2C_SPEED, CFG_I2C_SLAVE);
#endif
#ifdef CONFIG_LCD
    drv_lcd_init ();
#endif
#if defined(CONFIG_VIDEO) || defined(CONFIG_CFB_CONSOLE)
    drv_video_init ();
#endif
#ifdef CONFIG_KEYBOARD
    drv_keyboard_init ();
#endif
#ifdef CONFIG_LOGBUFFER
    drv_logbuff_init ();
#endif
    drv_system_init ();
#ifdef CONFIG_SERIAL_MULTI
    serial_devices_init ();
#endif
#ifdef CONFIG_USB_TTY
    drv_usbtty_init ();
#endif
#ifdef CONFIG_NETCONSOLE
    drv_nc_init ();
#endif
    return (0);
}
```

函数 drv_system_init ()总是要执行的，这个函数创建并注册了一个串行口设备和一个空设备（空设备是可选的），在 common/devices.c 中定义：

[common/devices.c]

```
static void drv_system_init (void)
{
    /* 创建串行口设备*/
    device_t dev;

    memset (&dev, 0, sizeof (dev));
    strcpy (dev.name, "serial");
    dev.flags = DEV_FLAGS_OUTPUT | DEV_FLAGS_INPUT | DEV_FLAGS_SYSTEM;
```

```
#ifdef CONFIG_SERIAL_SOFTWARE_FIFO
        dev.putc = serial_buffered_putc;
        dev.puts = serial_buffered_puts;
        dev.getc = serial_buffered_getc;
        dev.tstc = serial_buffered_tstc;
#else
        dev.putc = serial_putc;
        dev.puts = serial_puts;
        dev.getc = serial_getc;
        dev.tstc = serial_tstc;
#endif

    /* 注册设备：把设备加到设备链表 dev_list 中    */
        device_register (&dev);

    /* 根据需要创建空（NULL）设备*/
#ifdef CFG_DEVICE_NULLDEV
        memset (&dev, 0, sizeof (dev));

        strcpy (dev.name, "nulldev");
        dev.flags = DEV_FLAGS_OUTPUT | DEV_FLAGS_INPUT | DEV_FLAGS_SYSTEM;
        dev.putc = nulldev_putc;
        dev.puts = nulldev_puts;
        dev.getc = nulldev_input;
        dev.tstc = nulldev_input;

/* 注册设备：把设备加到设备链表 dev_list 中    */
        device_register (&dev);
#endif
}
```

3.7 控制台初始化

控制台初始化分两个阶段：console_init_f()和 console_init_r()。console_init_f()完成的功能很简单，只是根据环境变量设置了 global_data 中的一些数据成分(hasconsole， flag); console_init_r()在 common/console.c 中定义，完成主要的控制台初始化工作：在设备链表中搜索 stdin,stdout,stderr 设备；将搜索结果分别设为置控制台的 in, out 和 err 设备。

[common/console.c]

```
int console_init_r (void)
{
        DECLARE_GLOBAL_DATA_PTR;
        char *stdinname, *stdoutname, *stderrname;
        device_t *inputdev = NULL, *outputdev = NULL, *errdev = NULL;
#ifdef CFG_CONSOLE_ENV_OVERWRITE
```

```
        int i;
#endif /* CFG_CONSOLE_ENV_OVERWRITE */

        /*  设置跳转表   */
        gd->jt[XF_getc] = serial_getc;
        gd->jt[XF_tstc] = serial_tstc;
        gd->jt[XF_putc] = serial_putc;
        gd->jt[XF_puts] = serial_puts;
        gd->jt[XF_printf] = serial_printf;

        /*  从环境变量中读取 stdin,stdout,stderr 设备的名称   */
        stdinname    = getenv ("stdin");
        stdoutname = getenv ("stdout");
        stderrname = getenv ("stderr");

          /*  在设备列表中按名称搜索 stdin,stdout,stderr 设备，如果搜索不到，则设为默认的
          串行口设备  */
        if (OVERWRITE_CONSOLE == 0) {   /* if not overwritten by config switch */
                inputdev    = search_device (DEV_FLAGS_INPUT,    stdinname);
                outputdev = search_device (DEV_FLAGS_OUTPUT, stdoutname);
                errdev       = search_device (DEV_FLAGS_OUTPUT, stderrname);
        }
        /* if the devices are overwritten or not found, use default device */
        if (inputdev == NULL) {
                inputdev    = search_device (DEV_FLAGS_INPUT,    "serial");
        }
        if (outputdev == NULL) {
                outputdev = search_device (DEV_FLAGS_OUTPUT, "serial");
        }
        if (errdev == NULL) {
                errdev       = search_device (DEV_FLAGS_OUTPUT, "serial");
        }
        /*  设置控制台的标准输出设备   */
        /* Initializes output console first */
        if (outputdev != NULL) {
                console_setfile (stdout, outputdev);
        }
        /*  设置控制台的错误输出设备   */
        if (errdev != NULL) {
                console_setfile (stderr, errdev);
        }
        /*  设置控制台的标准输入设备   */
        if (inputdev != NULL) {
                console_setfile (stdin, inputdev);
```

```
        }

        gd->flags |= GD_FLG_DEVINIT; /* device initialization completed */

        /*  显示 in,out,err 设备的名称    */
#ifndef CFG_CONSOLE_INFO_QUIET
        /* Print information */
        puts ("In:       ");
        if (stdio_devices[stdin] == NULL) {
                puts ("No input devices available!\n");
        } else {
                printf ("%s\n", stdio_devices[stdin]->name);
        }

        puts ("Out:      ");
        if (stdio_devices[stdout] == NULL) {
                puts ("No output devices available!\n");
        } else {
                printf ("%s\n", stdio_devices[stdout]->name);
        }

        puts ("Err:      ");
        if (stdio_devices[stderr] == NULL) {
                puts ("No error devices available!\n");
        } else {
                printf ("%s\n", stdio_devices[stderr]->name);
        }
#endif /* CFG_CONSOLE_INFO_QUIET */

        /*  将 in,out,err 设备的名称保存到环境变量中  */
#ifdef CFG_CONSOLE_ENV_OVERWRITE
        /* set the environment variables (will overwrite previous env settings) */
        for (i = 0; i < 3; i++) {
                setenv (stdio_names[i], stdio_devices[i]->name);
        }
#endif /* CFG_CONSOLE_ENV_OVERWRITE */

#if 0
        /* If nothing usable installed, use only the initial console */
        if ((stdio_devices[stdin] == NULL) && (stdio_devices[stdout] == NULL))
                return (0);
#endif
        return (0);
}
```

```
static int console_setfile (int file, device_t * dev)
{
	DECLARE_GLOBAL_DATA_PTR;
	int error = 0;

	if (dev == NULL)
		return -1;

	switch (file) {
	case stdin:
	case stdout:
	case stderr:
		/* Start new device */
		if (dev->start) {
			error = dev->start ();
			/* If it's not started dont use it */
			if (error < 0)
				break;
		}

		/* Assign the new device (leaving the existing one started) */
		stdio_devices[file] = dev;

		/*
		 * Update monitor functions
		 * (to use the console stuff by other applications)
		 */
		switch (file) {
		case stdin:
			gd->jt[XF_getc] = dev->getc;
			gd->jt[XF_tstc] = dev->tstc;
			break;
		case stdout:
			gd->jt[XF_putc] = dev->putc;
			gd->jt[XF_puts] = dev->puts;
			gd->jt[XF_printf] = printf;
			break;
		}
		break;

	default:		/* Invalid file ID */
		error = -1;
	}
```

```
    return error;
}
```

通过 console_setfile()函数可以看出，控制台有一个包含 3 个 device_t 元素的数组 stdio_devices，分别对应 stdin,stdout,stderr。通过 stdio_devices[file] = dev 就可以将 dev 设成设置控制台的某个设备。这样就实现了控制台任意选择设备的功能。这和 linux 的设计思想有点类似。

3.8 单板后期初始化

函数 board_late_init()完成单板后期的初始化，对于 AT91SAM9260EK，这个函数在 board/at91sam9260ek/at91sam9260ek.c 中定义，调用 cpu/arm926ejs/at91sam9260/at91_emac.c 中的函数 eth_init()完成以太网的初始化。

[cpu/arm926ejs/at91sam9260/at91_emac.c]

```
int eth_init (bd_t * bd)
{
    unsigned int periphAEnable, periphBEnable;
    unsigned int val, i;
    int ret;

    p_mac = AT91C_BASE_EMACB;

    /* 配置 MAC 控制器的管脚 */
#ifdef CONFIG_AT91C_USE_RMII
    periphAEnable = ((unsigned int) AT91C_PA21_EMDIO    ) |
        ((unsigned int) AT91C_PA20_EMDC      ) |
        ((unsigned int) AT91C_PA19_ETXCK     ) |
        ((unsigned int) AT91C_PA18_ERXER     ) |
        ((unsigned int) AT91C_PA14_ERX0      ) |
        ((unsigned int) AT91C_PA17_ERXDV     ) |
        ((unsigned int) AT91C_PA15_ERX1      ) |
        ((unsigned int) AT91C_PA16_ETXEN     ) |
        ((unsigned int) AT91C_PA12_ETX0      ) |
        ((unsigned int) AT91C_PA13_ETX1      );

    periphBEnable = 0;
#else
    periphAEnable = ((unsigned int) AT91C_PA21_EMDIO    ) |
        ((unsigned int) AT91C_PA19_ETXCK     ) |
        ((unsigned int) AT91C_PA20_EMDC      ) |
        ((unsigned int) AT91C_PA18_ERXER     ) |
        ((unsigned int) AT91C_PA14_ERX0      ) |
        ((unsigned int) AT91C_PA17_ERXDV     ) |
        ((unsigned int) AT91C_PA15_ERX1      ) |
```

```
            ((unsigned int) AT91C_PA16_ETXEN      ) |
            ((unsigned int) AT91C_PA12_ETX0       ) |
            ((unsigned int) AT91C_PA13_ETX1       );

        periphBEnable = ((unsigned int) AT91C_PA27_ERXCK      ) |
            ((unsigned int) AT91C_PA29_ECOL       ) |
            ((unsigned int) AT91C_PA25_ERX2       ) |
            ((unsigned int) AT91C_PA26_ERX3       ) |
            ((unsigned int) AT91C_PA22_ETXER      ) |
            ((unsigned int) AT91C_PA10_ETX2       ) |
            ((unsigned int) AT91C_PA11_ETX3       ) |
            ((unsigned int) AT91C_PA28_ECRS         );
#endif
        AT91C_BASE_PIOA->PIO_ASR = periphAEnable;
        AT91C_BASE_PIOA->PIO_BSR = periphBEnable;
        AT91C_BASE_PIOA->PIO_PDR = (periphAEnable | periphBEnable);

        /* 禁止收发，清除收发状态寄存器 */
        p_mac->EMAC_NCR    =    0;    /*    ~(AT91C_EMAC_TE    |    AT91C_EMAC_RE    |
AT91C_EMAC_TSTART); */
        p_mac->EMAC_TSR = 0xFFFFFFFF;
        p_mac->EMAC_RSR = 0xFFFFFFFF;

        /* 打开 MAC 模块的时钟 */
        *AT91C_PMC_PCER = 1 << AT91C_ID_EMAC;      /* Peripheral Clock Enable Register */

        /* 禁止 PA17（RXDV） 的上拉电阻 */
        AT91C_BASE_PIOA->PIO_PPUDR = 1 << 17;

         /* 选择 MAC 接口为 MII 模式 */
         p_mac->EMAC_USRIO = AT91C_EMAC_CLKEN;

#ifdef CONFIG_AT91C_USE_RMII
        p_mac->EMAC_USRIO |= AT91C_EMAC_RMII;
#endif

        /* Init Ethernet buffers */
        RxBuffIndex = 0;
        TxBuffIndex = 0;

        /* 初始化接收缓冲区描述符链表 */
        for (i = 0; i < RBF_FRAMEMAX; ++i) {
            val = (unsigned int)(rbf_framebuf[i]);
            RxtdList[i].addr = val & 0xFFFFFFF8;
```

```
        RxtdList[i].U_Status.status = 0;
    }
    /* Set the WRAP bit at the end of the list descriptor */
    RxtdList[RBF_FRAMEMAX-1].addr |= RBF_WRAP;

    /* 初始化发送缓冲区描述符链表 */
    for (i = 0; i < TBF_FRAMEMAX; ++i) {
        val = (unsigned int)(tbf_framebuf[i]);
        TxtdList[i].addr = val & 0xFFFFFFF8;
        TxtdList[i].U_Status.status = 0;
        TxtdList[i].U_Status.S_Status.BuffUsed = 1;
    }
    TxtdList[0].U_Status.S_Status.BuffUsed = 0;
    /* Set the WRAP bit at the end of the list descriptor */
    TxtdList[TBF_FRAMEMAX-1].U_Status.S_Status.Wrap = 1;

    /* 获取 PHY 芯片的操作函数 */
    at91sam9260_GetPhyInterface (&PhyOps);

    /* 判断 PHY 芯片是否正常连接 */
    if (!PhyOps.IsPhyConnected (p_mac))
    {
        printf ("PHY not connected!\n\r");
        return -1;
    }
    else
        printf ("PHY is connected!\n\r");

    /* 调用 PHY 芯片的初始化函数：读取 PHY 芯片的工作模式，把 MAC 的工作模式设为和 PHY 一致 */
    /* MII management start from here */
    ret = PhyOps.Init (p_mac);
    if ( !ret && 0 )
    {
        printf ("MAC: error during MAC initialization\n");
        return -1;
    }

    /* MAC 初始化 */
    AT91F_EMACInit(bd, (unsigned int)RxtdList, (unsigned int)TxtdList);

    return 0;
}
```

```
int AT91F_EMACInit(bd_t * bd,
                   unsigned int pRxTdList,
                   unsigned int pTxTdList)
{
    unsigned int tick = 0;
    unsigned short status;

    /* 等待 PHY 芯片自协商完成 */
    at91sam9260_EmacEnableMDIO(p_mac);

    do {
        at91sam9260_EmacReadPhy(p_mac, AT91C_PHY_ADDR, MII_BMSR, &status);
        at91sam9260_EmacReadPhy(p_mac, AT91C_PHY_ADDR, MII_BMSR, &status);

        tick++;
    }
    while (!(status & BMSR_ANEGCOMPLETE) && (tick < AT91C_ETH_TIMEOUT));
    at91sam9260_EmacDisableMDIO(p_mac);
    printf ("End of Autonegociation\n\r");

    /* 设置 MAC 地址 */
    /* the sequence write EMAC_SA1L and write EMAC_SA1H must be respected */
    p_mac->EMAC_SA1L = (bd->bi_enetaddr[3] << 24) | (bd->bi_enetaddr[2] << 16)
                | (bd->bi_enetaddr[1] <<  8) | (bd->bi_enetaddr[0]);
    p_mac->EMAC_SA1H = (bd->bi_enetaddr[5] <<  8) | (bd->bi_enetaddr[4]);

    /* 设置接收、发送缓冲区队列头指针 */
    p_mac->EMAC_RBQP = pRxTdList;
    p_mac->EMAC_TBQP = pTxTdList;

    /* 清除接收状态寄存器相关位 */
    p_mac->EMAC_RSR        &=    ~(AT91C_EMAC_OVR    |    AT91C_EMAC_REC    |
AT91C_EMAC_BNA);

    /* 设置配置寄存器：拷贝所有帧，不支持广播  */
    p_mac->EMAC_NCFGR   |= (AT91C_EMAC_CAF | AT91C_EMAC_NBC );
    p_mac->EMAC_NCFGR   &= ~(AT91C_EMAC_CLK);

    /* 设置 PHY 管理接口的时钟速率，不能超过 2.5MHz  */
#if (AT91C_MASTER_CLOCK > 40000000)
    /* MDIO clock must not exceed 2.5 MHz, so enable MCK divider */
    p_mac->EMAC_NCFGR |= AT91C_EMAC_CLK_HCLK_64;
#endif
    /* 设置控制寄存器：接收允许，发送允许，允许统计功能 */
```

```
p_mac->EMAC_NCR |= (AT91C_EMAC_TE | AT91C_EMAC_RE |
AT91C_EMAC_WESTAT);

    return 0;
}
```

4 命令处理

4.1 命令数据结构

U-boot 的命令用 struct cmd_tbl_t 来实现。cmd_tbl_t 的主要数据成分是命令名称(name)和命令处理函数(cmd)，此外还包括最大参数个数(maxargs)，是否可重复执行(repeatable)，使用方法和帮助信息(usage,help)等。这个数据结构在文件 include/command.h 中定义：

[include/command.h]

```
struct cmd_tbl_s {
    char    *name;      /* Command Name             */
    int     maxargs;    /* maximum number of arguments */
    int     repeatable; /* autorepeat allowed?      */
                        /* Implementation function  */
    int     (*cmd)(struct cmd_tbl_s *, int, int, char *[]);
    char    *usage;     /* Usage message  (short)   */
#ifdef CFG_LONGHELP
    char    *help;      /* Help  message  (long)    */
#endif
#ifdef CONFIG_AUTO_COMPLETE
    /* do auto completion on the arguments */
    int     (*complete)(int argc, char *argv[], char last_char, int maxv, char *cmdv[]);
#endif
};
typedef struct cmd_tbl_s cmd_tbl_t;
```

包含在 include/command.h 中的宏 U_BOOT_CMD 用来定义命令：

```
#define Struct_Section  __attribute__ ((unused,section (".u_boot_cmd")))
#define U_BOOT_CMD(name,maxargs,rep,cmd,usage,help) \
cmd_tbl_t __u_boot_cmd_##name Struct_Section = {#name, maxargs, rep, cmd, usage, help}
```

例如 bootm 命令的定义：

```
U_BOOT_CMD(
    bootm,  CFG_MAXARGS, 1,  do_bootm,
    "bootm    - boot application image from memory\n",
    "[addr [arg ...]]\n    - boot application image stored in memory\n"
```

```
        "\tpassing arguments 'arg ...'; when booting a Linux kernel,\n"
        "\t'arg' can be the address of an initrd image\n"
);
```

展开后就变成：

```
cmd_tbl_t   __u_boot_cmd_bootm   __attribute__ ((unused,section (".u_boot_cmd"))) =
{
   “bootm”,   /* name */
     CFG_MAX_ARGS, /* maxargs */
     1,     /* repeatable */
     do_bootm,   /* cmd   */
     "bootm      - boot application image from memory\n", /*   usage   */
      "[addr [arg ...]]\n       - boot application image stored in memory\n"
      "\tpassing arguments 'arg ...'; when booting a Linux kernel,\n"
      "\t'arg' can be the address of an initrd image\n"    /* help */
};
```

这样就为 bootm 命令定义了一个 cmd_tbl_t 结构。注意定义的结构是放在”.u_boot.cmd”段中的，这是为了实现命令的查找功能。

4.2 命令查找

要想执行命令，必须先找到这个命令对应的 cmd_tbl_t，然后才能调用命令处理函数。文件 common/command.c 中的 find_command()用来查找命令。这个函数根据命令名称来搜索命令列表，返回命令的 cmd_tbl_t 指针。

[common/command.c]

```
cmd_tbl_t *find_cmd (const char *cmd)
{
      cmd_tbl_t *cmdtp;
      cmd_tbl_t *cmdtp_temp = &__u_boot_cmd_start;      /*Init value */
      const char *p;
      int len;
      int n_found = 0;

      /*   需要比较的命令名称长度，只比较”.”之前的字符串   */
      len = ((p = strchr(cmd, '.')) == NULL) ? strlen (cmd) : (p - cmd);

      /*   搜索命令列表，如果命令名称匹配，返回 cmd_tbl_t 指针   */
      for (cmdtp = &__u_boot_cmd_start;
           cmdtp != &__u_boot_cmd_end;
           cmdtp++) {
          if (strncmp (cmd, cmdtp->name, len) == 0) {
              if (len == strlen (cmdtp->name))
                  return cmdtp;  /* full match */
```

```
                cmdtp_temp = cmdtp;    /* abbreviated command ? */
                n_found++;
            }
        }
        if (n_found == 1) {             /* exactly one match */
            return cmdtp_temp;
        }

        return NULL;  /* not found or ambiguous command */
}
```

注意其中的&__u_boot_cmd_start 和&__u_boot_cmd_end，这是命令列表的起始和结束地址。在链接脚本 board/at91sam9260ek/u-boot.lds 中可以找到：

```
__u_boot_cmd_start = .;
.u_boot_cmd : { *(.u_boot_cmd) }
__u_boot_cmd_end = .;
```

由于所有的命令结构都是放在".u_boot.cmd"段中的（参见 4.1），因此通过这种 方法就构建了一个所有命令结构的列表。其实这种方法和 linux 中对初始化函数的处理方法是一样的。

4.3 主循环

主循环的实现函数 main_loop()在 common/main.c 中。这个函数包含了很多与 AT91SAM9260EK 关系不大的条件编译语句，下面列出的代码做了一些精简。

[common/main.c]

```
void main_loop (void)
{
#ifndef CFG_HUSH_PARSER
        static char lastcommand[CFG_CBSIZE] = { 0, };
        int len;
        int rc = 1;
        int flag;
#endif

#if defined(CONFIG_BOOTDELAY) && (CONFIG_BOOTDELAY >= 0)
        char *s;
        int bootdelay;
#endif

#ifdef CONFIG_PREBOOT
        char *p;
#endif
#ifdef CONFIG_BOOTCOUNT_LIMIT
```

```
        unsigned long bootcount = 0;
        unsigned long bootlimit = 0;
        char *bcs;
        char bcs_set[16];
#endif /* CONFIG_BOOTCOUNT_LIMIT */

/* 如果允许“引导次数限制”功能，读入引导计数，加 1，再保存起来； 并从环境变量读取引导次数上限 bootlimit */
#ifdef CONFIG_BOOTCOUNT_LIMIT
        bootcount = bootcount_load();
        bootcount++;
        bootcount_store (bootcount);
        sprintf (bcs_set, "%lu", bootcount);
        setenv ("bootcount", bcs_set);
        bcs = getenv ("bootlimit");
        bootlimit = bcs ? simple_strtoul (bcs, NULL, 10) : 0;
#endif /* CONFIG_BOOTCOUNT_LIMIT */

/* 如果允许“PREBOOT”功能，从环境变量读取 preboot 命令序列，并执行 */
#ifdef CONFIG_PREBOOT
        if ((p = getenv ("preboot")) != NULL) {
# ifdef CONFIG_AUTOBOOT_KEYED
                int prev = disable_ctrlc(1);    /* disable Control C checking */
# endif
                run_command (p, 0);

# ifdef CONFIG_AUTOBOOT_KEYED
                disable_ctrlc(prev); /* restore Control C checking */
# endif
        }
#endif /* CONFIG_PREBOOT */

/* 如果定义了 CONFIG_BOOTDELAY，从环境变量读取 bootdelay  */
#if defined(CONFIG_BOOTDELAY) && (CONFIG_BOOTDELAY >= 0)
        s = getenv ("bootdelay");
        bootdelay = s ? (int)simple_strtol(s, NULL, 10) : CONFIG_BOOTDELAY;

        debug ("### main_loop entered: bootdelay=%d\n\n", bootdelay);
/*
检查引导次数是否已超过指定的上限 bootlimit。如果超出上限，从环境变量中读取 altbootcmd 命令序列到变量 p；否则读取 bootcmd 命令序列到 p
*/
#ifdef CONFIG_BOOTCOUNT_LIMIT
        if (bootlimit && (bootcount > bootlimit)) {
```

```
            printf ("Warning: Bootlimit (%u) exceeded. Using altbootcmd.\n",
                    (unsigned)bootlimit);
            s = getenv ("altbootcmd");
        }
        else
#endif /* CONFIG_BOOTCOUNT_LIMIT */
            s = getenv ("bootcmd");

        debug ("### main_loop: bootcmd=\"%s\"\n", s ? s : "<UNDEFINED>");

/* 检查在制定的 bootdelay 期限内，用户是否选择了终止引导过程  */
        if (bootdelay >= 0 && s && !abortboot (bootdelay)) {
# ifdef CONFIG_AUTOBOOT_KEYED
            int prev = disable_ctrlc(1);    /* disable Control C checking */
# endif
/* 用户未终止引导，执行命令 bootcmd （或 altbootcmd）命令序列，引导指定操作系统，
不再返回  */
            run_command (s, 0);

# ifdef CONFIG_AUTOBOOT_KEYED
            disable_ctrlc(prev); /* restore Control C checking */
# endif
        }

#endif    /* CONFIG_BOOTDELAY */

/* 用户终止了引导，进入命令循环  */
        for (;;) {
            len = readline (CFG_PROMPT);    /* 从控制台读取命令 */
            flag = 0;  /* assume no special flags for now */
            /* 如果命令长度>0，拷贝到 lastcommand */
            if (len > 0)
                strcpy (lastcommand, console_buffer);
            else if (len == 0)
            /* 否则，设置“重复执行命令”标记 */
                flag |= CMD_FLAG_REPEAT;

            if (len == -1)
                puts ("<INTERRUPT>\n");
            else
                rc = run_command (lastcommand, flag);    /* 执行命令 */

            /* 如果命令执行失败或者命令不可重复执行，清除 lastcommand  */
```

```
            if (rc <= 0) {
                /* invalid command or not repeatable, forget it */
                lastcommand[0] = 0;
            }
        }
}
```

其中最重要的函数有两个：abortboot ()和 run_command()，我们分开来介绍。

(1) abortboot ()

这个函数用来判断在指定的时间期限内用户是否选择了中止系统的引导。函数的定义也在 common/main.c 中。下面列出的代码作了一些精简。

[common/main.c]

```
static __inline__ int abortboot(int bootdelay)
{
    int abort = 0;

/* 如果定义了 CONFIG_SLIENT_CONSOLE，把控制台的 stdout，stderr 设备指向串行口设备 */
    #ifdef CONFIG_SILENT_CONSOLE
    {
        DECLARE_GLOBAL_DATA_PTR;

        if (gd->flags & GD_FLG_SILENT) {
            /* Restore serial console */
            console_assign (stdout, "serial");
            console_assign (stderr, "serial");
        }
    }
#endif

/* 显示信息：Hit any key to stop autoboot: … */
#ifdef CONFIG_MENUPROMPT
    printf(CONFIG_MENUPROMPT, bootdelay);
#else
    printf("Hit any key to stop autoboot: %2d ", bootdelay);
#endif

/* 如果定义了 CONFIG_ZERO_BOOTDELAY_CHECK，先检查有没有键被按下 */
#if defined CONFIG_ZERO_BOOTDELAY_CHECK
    /*
     * Check if key already pressed
```

```
         * Don't check if bootdelay < 0
         */
        if (bootdelay >= 0) {
                if (tstc()) {      /* we got a key press      */
                        (void) getc();    /* consume input   */
                        puts ("\b\b\b 0");
                        abort = 1;        /* don't auto boot    */
                }
        }
#endif

/*  检查是否有键被按下的循环   */
        while ((bootdelay > 0) && (!abort)) {    /*  延时未到  &  未选择终止  */
                int i;
                --bootdelay;    /*  延时计数减 1（1 秒钟）*/
/*  延时 100*10ms = 1s */
                for (i=0; !abort && i<100; ++i) {
                        if (tstc()) {              /*  有键被按下          */
                                abort    = 1;      /*  终止引导       */
                                bootdelay = 0;   /*    延时计数清零      */
                                (void) getc();    /*  读按键（清输入缓冲区，以便下次正确调用 tstc() */
                                break;
                        }
                        udelay (10000);
                }

                printf ("\b\b\b%2d ", bootdelay);    /*  显示剩余时间   */
        }

        putc ('\n');

/*  如果定义了  CONFIG_SLIENT_CONSOLE，又未选择终止引导，把控制台的  stdout，
stderr 设备指向空设备（nulldev）；否则清除 GD_FLG_SILENT  标记  */
#ifdef CONFIG_SILENT_CONSOLE
        {
                DECLARE_GLOBAL_DATA_PTR;
                if (abort) {
                        /* permanently enable normal console output */
                        gd->flags &= ~(GD_FLG_SILENT);
                } else if (gd->flags & GD_FLG_SILENT) {
                        /* Restore silent console */
                        console_assign (stdout, "nulldev");
                        console_assign (stderr, "nulldev");
                }
```

```
    }
#endif
    return abort;    /* 返回结果：1＝中止引导 */
}
```

(2) run_command()

这个函数用来执行命令。它可以执行一条命令，也可以执行用";"分隔的多条命令（例如"nand read 20400000 0 200000; nand read 21100000 200000 400000; bootm 20400000"）。run_command()也在 common/main.c 中定义。

[common/main.c]

```
int run_command (const char *cmd, int flag)
{
    cmd_tbl_t *cmdtp;
    char cmdbuf[CFG_CBSIZE];/* working copy of cmd        */
    char *token;            /* start of token in cmdbuf   */
    char *sep;              /* end of token (separator) in cmdbuf */
    char finaltoken[CFG_CBSIZE];
    char *str = cmdbuf;
    char *argv[CFG_MAXARGS + 1];    /* NULL terminated    */
    int argc, inquotes;
    int repeatable = 1;
    int rc = 0;

#ifdef DEBUG_PARSER
    printf ("[RUN_COMMAND] cmd[%p]=\"", cmd);
    puts (cmd ? cmd : "NULL");  /* use puts - string may be loooong */
    puts ("\"\n");
#endif

    clear_ctrlc();          /* forget any previous Control C */

    /* 参数有效性检查 */
    if (!cmd || !*cmd) {
        return -1; /* empty command */
    }

    if (strlen(cmd) >= CFG_CBSIZE) {
        puts ("## Command too long!\n");
        return -1;
    }

    strcpy (cmdbuf, cmd);
```

```
    /* Process separators and check for invalid
     * repeatable commands
     */

#ifdef DEBUG_PARSER
    printf ("[PROCESS_SEPARATORS] %s\n", cmd);
#endif
    /* 分隔命令循环：根据';'将命令序列分隔成多条命令，结果放入 token */
    while (*str) {
        /*
         * Find separator, or string end
         * Allow simple escape of ';' by writing "\;"
         */
        for (inquotes = 0, sep = str; *sep; sep++) {
            if ((*sep=='\"') &&
                (*(sep-1) != '\\'))
                inquotes=!inquotes;

            if (!inquotes &&
                (*sep == ';') &&     /* separator           */
                ( sep != str) &&     /* past string start   */
                (*(sep-1) != '\\'))  /* and NOT escaped     */
                break;
        }

        /*
         * Limit the token to data between separators
         */
        token = str;
        if (*sep) {
            str = sep + 1;   /* start of command for next pass */
            *sep = '\0';
        }
        else
            str = sep; /* no more commands for next pass */
#ifdef DEBUG_PARSER
        printf ("token: \"%s\"\n", token);
#endif
        /* 处理命令中的宏替换 */
        /* find macros in this token and replace them */
        process_macros (token, finaltoken);

        /* Extract arguments */
```

```
            /* 解析命令参数 */
            argc = parse_line (finaltoken, argv);

            /* 查找命令的 cmd_tbl_t 结构指针 */
            if ((cmdtp = find_cmd(argv[0])) == NULL) {
                printf ("Unknown command '%s' - try 'help'\n", argv[0]);
                rc = -1;    /* give up after bad command */
                continue;
            }

            /* 检查命令 maxargs */
            if (argc > cmdtp->maxargs) {
                printf ("Usage:\n%s\n", cmdtp->usage);
                rc = -1;
                continue;
            }
            /* bootd 命令处理 */
#if (CONFIG_COMMANDS & CFG_CMD_BOOTD)
            /* avoid "bootd" recursion */
            if (cmdtp->cmd == do_bootd) {
#ifdef DEBUG_PARSER
                printf ("[%s]\n", finaltoken);
#endif
                if (flag & CMD_FLAG_BOOTD) {
                    puts ("'bootd' recursion detected\n");
                    rc = -1;
                    continue;
                }
                else
                    flag |= CMD_FLAG_BOOTD;
            }
#endif    /* CFG_CMD_BOOTD */

           /* 调用命令的处理函数 */
            if ((cmdtp->cmd) (cmdtp, flag, argc, argv) != 0) {
                rc = -1;
            }
           /* 设置“可重复执行“标记 */
            repeatable &= cmdtp->repeatable;

            /* 如果用户强行终止，返回 0（已执行完当前命令，不再执行后续命令） */
            if (had_ctrlc ())
                return 0;  /* if stopped then not repeatable */
        }
```

```
        return rc ? rc : repeatable;    /* 返回“命令可重复执行”或－1  */
}
```

5 linux 的引导

5.1 映象格式

映象文件必须满足 U-boot 的格式要求，才能被识别和引导。U-boot 中映象文件必须以一个固定格式的头部开始。 这个头部由 struct image_header_t 描述，image_header_t 的定义在文件 include/image.h 中。

[include/image.h]

```
typedef struct image_header {
    uint32_t  ih_magic;/* Image Header Magic Number    */
    uint32_t  ih_hcrc;  /* Image Header CRC Checksum  */
    uint32_t  ih_time;  /* Image Creation Timestamp*/
    uint32_t  ih_size;  /* Image Data Size        */
    uint32_t  ih_load;  /* Data     Load   Address        */
    uint32_t  ih_ep;          /* Entry Point Address        */
    uint32_t  ih_dcrc;  /* Image Data CRC Checksum     */
    uint8_t         ih_os;          /* Operating System          */
    uint8_t         ih_arch;  /* CPU architecture      */
    uint8_t         ih_type;  /* Image Type            */
    uint8_t         ih_comp; /* Compression Type          */
    uint8_t         ih_name[IH_NMLEN];  /* Image Name            */
} image_header_t;
```

U-boot 以源代码的形式提供了一个映象文件制作工具 mkimage（在 tools 目录下），这个工具可以为指定的映象文件增加一个 image_header_t 头部。

5.2 linux 引导

通过前面的分析，我们知道，如果启动过程中用户不按键中止引导，命令序列 bootcmd 将会被执行。对于 AT91SAM9260eK 板，bootcmd 的内容是"nand read 20400000 0 200000;nand read 21100000 200000 400000;bootm 20400000"。这意味着将执行三条命令：

(1) nand read 20400000 0 200000

将 NAND Flash 中从 0 开始长度为 200000（2MB）的数据块读入地址 20400000。NAND Flash 中从 0 开始存放的是 linux 内核映象，20400000 是 SDRAM 的地址。 所以这条命令的功能是把 linux 内核映象从 NAND Flash 读到 SDRAM 中。注意这个映象是满足 Uboot 格式要求，即是以 image_header_t 头部开始的。

(2) nand read 21100000 200000 400000

将 NAND Flash 中从 200000 开始长度为 400000（4MB）的数据块读入地址 21100000。NAND Flash 中从 200000 开始存放的是 linux 根文件系统映象，21100000 是 SDRAM 的地址。 所以这条命令的功能是把 linux 根文件系统映象从 NAND Flash 读到 SDRAM 中。这个映象不需要满足 U-boot 的格式要求。

(3) bootm 20400000

执行 bootm 命令引导 linux。命令的参数是 linux 内核所在的地址 20400000。

由此可见，linux 的引导是通过 bootm 命令实现的。这个命令的处理函数是 do_bootm()，在文件 common/cmd_bootm.c 中。U-boot 可以引导多种操作系统。例如 linux，vxworks，netbsd，QNX 等等，下面列出的代码省去了和 linux 系统和 arm 平台无关的部分。

[common/cmd_bootm.c]

```
int do_bootm (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
    ulong    iflag;
    ulong    addr;
    ulong    data, len, checksum;
    ulong   *len_ptr;
    uint  unc_len = 0x400000;
    int    i, verify;
    char *name, *s;
    int    (*appl)(int, char *[]);
    image_header_t *hdr = &header;

  /*  从环境变量读取 verify :“是否校验”标记    */
    s = getenv ("verify");
    verify = (s && (*s == 'n')) ? 0 : 1;

  /*  从命令参数获取 addr，对于 AT91SAM9260EK，结果为 0x20400000    */
    if (argc < 2) {
        addr = load_addr;
    } else {
        addr = simple_strtoul(argv[1], NULL, 16);
    }

  /*  显示启动进度信息  */
    SHOW_BOOT_PROGRESS (1);
    printf ("## Booting image at %08lx ...\n", addr);

  /*  如果地址 addr 在 DataFlash 地址空间内，
```

```
    从 DataFlash 中读入映象文件头部到 header
    对于 AT91SAM9260EK，这段代码不会被执行，因为 addr = 0x20400000，
    是 SDRAM 空间，不是 DataFlash 空间
    */
#ifdef CONFIG_HAS_DATAFLASH
      if (addr_dataflash(addr)){
            read_dataflash(addr, sizeof(image_header_t), (char *)&header);
      } else
#endif
    /*   否则，从 addr 指定的位置读入映象文件头部到 header    */
      memmove (&header, (char *)addr, sizeof(image_header_t));

    /*   检查头部的 magic number，如果出错，返回 1    */
      if (ntohl(hdr->ih_magic) != IH_MAGIC) {
            puts ("Bad Magic Number\n");
            SHOW_BOOT_PROGRESS (-1);
            return 1;
      }
      SHOW_BOOT_PROGRESS (2);

    /*   计算并检查映象头部的 CRC，如果出错，返回 1    */
      data = (ulong)&header;
      len    = sizeof(image_header_t);
      checksum = ntohl(hdr->ih_hcrc);
      hdr->ih_hcrc = 0;
      if (crc32 (0, (uchar *)data, len) != checksum) {
            puts ("Bad Header Checksum\n");
            SHOW_BOOT_PROGRESS (-2);
            return 1;
      }
      SHOW_BOOT_PROGRESS (3);

    /*
       如果系统有 DataFlash，而且地址 addr 在 DataFlash 地址空间内，
       从 DataFlash 中读入整个映象文件到默认加载地址 CFG_LOAD_ADDR
       对于 AT91SAM9260EK，这段代码不会被执行。
    */
#ifdef CONFIG_HAS_DATAFLASH
      if (addr_dataflash(addr)){
            len    = ntohl(hdr->ih_size) + sizeof(image_header_t);
            read_dataflash(addr, len, (char *)CFG_LOAD_ADDR);
            addr = CFG_LOAD_ADDR;
      }
#endif
```

```
    /* 显示映象文件头部信息 */
    print_image_hdr ((image_header_t *)addr);

     /* 跳过映象文件头部   */
    data = addr + sizeof(image_header_t);
    len   = ntohl(hdr->ih_size);

    /* 计算并检查映象数据部分的 CRC，如果出错，返回 1    */
    if (verify) {
        puts ("      Verifying Checksum ... ");
        if (crc32 (0, (uchar *)data, len) != ntohl(hdr->ih_dcrc)) {
            printf ("Bad Data CRC\n");
            SHOW_BOOT_PROGRESS (-3);
            return 1;
        }
        puts ("OK\n");
    }
    SHOW_BOOT_PROGRESS (4);

    len_ptr = (ulong *)data;

    /* 检查机器类型，如果出错，返回 1    */
#if defined(__PPC__)
    if (hdr->ih_arch != IH_CPU_PPC)
#elif defined(__ARM__)
    if (hdr->ih_arch != IH_CPU_ARM)
…
…
#else
# error Unknown CPU type
#endif
    {
        printf ("Unsupported Architecture 0x%x\n", hdr->ih_arch);
        SHOW_BOOT_PROGRESS (-4);
        return 1;
    }
    SHOW_BOOT_PROGRESS (5);

    /* 判断映象类型    */
    switch (hdr->ih_type) {
    case IH_TYPE_STANDALONE:
        name = "Standalone Application";
        /* A second argument overwrites the load address */
        if (argc > 2) {
```

```
                hdr->ih_load = htonl(simple_strtoul(argv[2], NULL, 16));
            }
            break;
        case IH_TYPE_KERNEL:
            name = "Kernel Image";
            break;
        case IH_TYPE_MULTI:
            name = "Multi-File Image";
            len  = ntohl(len_ptr[0]);
            /* OS kernel is always the first image */
            data += 8; /* kernel_len + terminator */
            for (i=1; len_ptr[i]; ++i)
                data += 4;
            break;
        default: printf ("Wrong Image Type for %s command\n", cmdtp->name);
            SHOW_BOOT_PROGRESS (-5);
            return 1;
        }
        SHOW_BOOT_PROGRESS (6);

        /*
         * We have reached the point of no return: we are going to
         * overwrite all exception vector code, so we cannot easily
         * recover from any failures any more...
         */

        /* 关中断    */
        iflag = disable_interrupts();

        /* 判断映象压缩类型，如果没有压缩，拷贝映象数据到映象文件要求的加载地址
           hdr->ih_load; 否则根据压缩类型调用相应的解压缩函数将映象数据解压缩到
           hdr->ih_load
        */
        switch (hdr->ih_comp) {
        case IH_COMP_NONE:
            if(ntohl(hdr->ih_load) == addr) {
                printf ("    XIP %s ... ", name);
            } else {
#if defined(CONFIG_HW_WATCHDOG) || defined(CONFIG_WATCHDOG)
                size_t l = len;
                void *to = (void *)ntohl(hdr->ih_load);
                void *from = (void *)data;

                printf ("    Loading %s ... ", name);
```

```
                while (l > 0) {
                        size_t tail = (l > CHUNKSZ) ? CHUNKSZ : l;
                        WATCHDOG_RESET();
                        memmove (to, from, tail);
                        to += tail;
                        from += tail;
                        l -= tail;
                }
#else   /* !(CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG) */
                memmove ((void *) ntohl(hdr->ih_load), (uchar *)data, len);
#endif  /* CONFIG_HW_WATCHDOG || CONFIG_WATCHDOG */
        }
        break;
    case IH_COMP_GZIP:
        printf ("   Uncompressing %s ... ", name);
        if (gunzip ((void *)ntohl(hdr->ih_load), unc_len,
                        (uchar *)data, &len) != 0) {
                puts ("GUNZIP ERROR - must RESET board to recover\n");
                SHOW_BOOT_PROGRESS (-6);
                do_reset (cmdtp, flag, argc, argv);
        }
        break;
#ifdef CONFIG_BZIP2
    case IH_COMP_BZIP2:
        printf ("   Uncompressing %s ... ", name);
        /*
          * If we've got less than 4 MB of malloc() space,
          * use slower decompression algorithm which requires
          * at most 2300 KB of memory.
          */
        i = BZ2_bzBuffToBuffDecompress ((char*)ntohl(hdr->ih_load),
                                &unc_len, (char *)data, len,
                                CFG_MALLOC_LEN < (4096 * 1024), 0);
        if (i != BZ_OK) {
                printf ("BUNZIP2 ERROR %d - must RESET board to recover\n", i);
                SHOW_BOOT_PROGRESS (-6);
                udelay(100000);
                do_reset (cmdtp, flag, argc, argv);
        }
        break;
#endif /* CONFIG_BZIP2 */
    default:
        if (iflag)
```

```
            enable_interrupts();
        printf ("Unimplemented compression type %d\n", hdr->ih_comp);
        SHOW_BOOT_PROGRESS (-7);
        return 1;
    }
    puts ("OK\n");
    SHOW_BOOT_PROGRESS (7);

    /* 判断映象类型，对于 STANDALONE 映象（独立运行的程序），开中断，调用映象
       程序，然后返回 0；其它类型的映象什么也不做
    */
    switch (hdr->ih_type) {
    case IH_TYPE_STANDALONE:
        if (iflag)
            enable_interrupts();

        /* load (and uncompress), but don't start if "autostart"
         * is set to "no"
         */
        if (((s = getenv("autostart")) != NULL) && (strcmp(s,"no") == 0)) {
            char buf[32];
            sprintf(buf, "%lX", len);
            setenv("filesize", buf);
            return 0;
        }
        appl = (int (*)(int, char *[]))ntohl(hdr->ih_ep);
        (*appl)(argc-1, &argv[1]);
        return 0;
    case IH_TYPE_KERNEL:
    case IH_TYPE_MULTI:
        /* handled below */
        break;
    default:
        if (iflag)
            enable_interrupts();
        printf ("Can't boot image type %d\n", hdr->ih_type);
        SHOW_BOOT_PROGRESS (-8);
        return 1;
    }
    SHOW_BOOT_PROGRESS (8);

    /* 根据映象的操作系统类型，调用相应的函数完成引导。 对于 linux，调用的是
       do_bootm_linux()函数
    */
```

```
        switch (hdr->ih_os) {
        default:                /* handled by (original) Linux case */
        case IH_OS_LINUX:
#ifdef CONFIG_SILENT_CONSOLE
                fixup_silent_linux();
#endif
                do_bootm_linux    (cmdtp, flag, argc, argv,
                                   addr, len_ptr, verify);
                break;
        case IH_OS_NETBSD:
                do_bootm_netbsd (cmdtp, flag, argc, argv,
                                 addr, len_ptr, verify);
                break;

#ifdef CONFIG_LYNXKDI
        case IH_OS_LYNXOS:
                do_bootm_lynxkdi (cmdtp, flag, argc, argv,
                                  addr, len_ptr, verify);
                break;
#endif
…
…
        }

        /* 以下代码在正常情况下不会执行到，因为引导函数不会返回     */
      SHOW_BOOT_PROGRESS (-9);
#ifdef DEBUG
        puts ("\n## Control returned to monitor - resetting...\n");
        do_reset (cmdtp, flag, argc, argv);
#endif
        return 1;
}
```

下面看 linux 引导的第二阶段 do_bootm_linux()，这个函数在 lib_arm/armlinux.c 中。

[lib_arm/armlinux.c]

```
void do_bootm_linux (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[],
                     ulong addr, ulong *len_ptr, int verify)
{
        DECLARE_GLOBAL_DATA_PTR;

        ulong len = 0, checksum;
        ulong initrd_start, initrd_end;
```

```
    ulong data;
    void (*theKernel)(int zero, int arch, uint params);
    image_header_t *hdr = &header;
    bd_t *bd = gd->bd;

    /*
     从环境变量 bootargs 中读取命令行到 commandline。 对于 AT91SAM9260EK,
     为: "mem=64M  console=ttyS0,115200  initrd=0x21100000,17000000  root=/dev/ram0
       rw"
    */
#ifdef CONFIG_CMDLINE_TAG
    char *commandline = getenv ("bootargs");
#endif
    /*   theKernel 指向内核入口  */
    theKernel = (void (*)(int, int, uint))ntohl(hdr->ih_ep);

    /*   检查是否在 bootm 命令中提供了根文件系统映象的地址，如果是，处理映象。
       处理过程和对内核映象的处理基本一致，包括检查 magic number,计算并检查头部
       CRC，数据 CRC 等等。对于 AT91SAM9260EK，这段代码不会被执行，因为在
       bootm 命令中只指定了 linux 内核映象的地址，根文件系统映象的地址是通过命令
       行信息 commandline 传递给内核的。
    */

    /*
     * Check if there is an initrd image
     */
    if (argc >= 3) {
        SHOW_BOOT_PROGRESS (9);

        addr = simple_strtoul (argv[2], NULL, 16);

        printf ("## Loading Ramdisk Image at %08lx ...\n", addr);

        /* Copy header so we can blank CRC field for re-calculation */
#ifdef CONFIG_HAS_DATAFLASH
        if (addr_dataflash (addr)) {
            read_dataflash (addr, sizeof (image_header_t),
                    (char *) &header);
        } else
#endif
            memcpy (&header, (char *) addr,
                sizeof (image_header_t));

        if (ntohl (hdr->ih_magic) != IH_MAGIC) {
```

```
                printf ("Bad Magic Number\n");
                SHOW_BOOT_PROGRESS (-10);
                do_reset (cmdtp, flag, argc, argv);
        }

        data = (ulong) & header;
        len = sizeof (image_header_t);

        checksum = ntohl (hdr->ih_hcrc);
        hdr->ih_hcrc = 0;

        if (crc32 (0, (char *) data, len) != checksum) {
                printf ("Bad Header Checksum\n");
                SHOW_BOOT_PROGRESS (-11);
                do_reset (cmdtp, flag, argc, argv);
        }

        SHOW_BOOT_PROGRESS (10);

        print_image_hdr (hdr);

        data = addr + sizeof (image_header_t);
        len = ntohl (hdr->ih_size);

#ifdef CONFIG_HAS_DATAFLASH
        if (addr_dataflash (addr)) {
                read_dataflash (data, len, (char *) CFG_LOAD_ADDR);
                data = CFG_LOAD_ADDR;
        }
#endif

        if (verify) {
                ulong csum = 0;

                printf ("      Verifying Checksum ... ");
                csum = crc32 (0, (char *) data, len);
                if (csum != ntohl (hdr->ih_dcrc)) {
                        printf ("Bad Data CRC\n");
                        SHOW_BOOT_PROGRESS (-12);
                        do_reset (cmdtp, flag, argc, argv);
                }
                printf ("OK\n");
        }
```

```
            SHOW_BOOT_PROGRESS (11);

            if ((hdr->ih_os != IH_OS_LINUX) ||
                (hdr->ih_arch != IH_CPU_ARM) ||
                (hdr->ih_type != IH_TYPE_RAMDISK)) {
                printf ("No Linux ARM Ramdisk Image\n");
                SHOW_BOOT_PROGRESS (-13);
                do_reset (cmdtp, flag, argc, argv);
            }

#if defined(CONFIG_B2) || defined(CONFIG_EVB4510) || defined(CONFIG_ARMADILLO)
            /*
              *we need to copy the ramdisk to SRAM to let Linux boot
              */
            memmove ((void *) ntohl(hdr->ih_load), (uchar *)data, len);
            data = ntohl(hdr->ih_load);
#endif /* CONFIG_B2 || CONFIG_EVB4510 */

            /*
              * Now check if we have a multifile image
              */
        } else if ((hdr->ih_type == IH_TYPE_MULTI) && (len_ptr[1])) {
            ulong tail = ntohl (len_ptr[0]) % 4;
            int i;

            SHOW_BOOT_PROGRESS (13);

            /* skip kernel length and terminator */
            data = (ulong) (&len_ptr[2]);
            /* skip any additional image length fields */
            for (i = 1; len_ptr[i]; ++i)
                data += 4;
            /* add kernel length, and align */
            data += ntohl (len_ptr[0]);
            if (tail) {
                data += 4 - tail;
            }

            len = ntohl (len_ptr[1]);

        } else {
            /*
              * no initrd image
              */
```

```
            SHOW_BOOT_PROGRESS (14);

            len = data = 0;
        }

#ifdef      DEBUG
        if (!data) {
            printf ("No initrd\n");
        }
#endif

        if (data) {
            initrd_start = data;
            initrd_end = initrd_start + len;
        } else {
            initrd_start = 0;
            initrd_end = 0;
        }

        SHOW_BOOT_PROGRESS (15);

        debug ("## Transferring control to Linux (at address %08lx) ...\n",
                (ulong) theKernel);

        /*    设置传递给 linux 内核的参数表 ：tagged list    */
#if defined (CONFIG_SETUP_MEMORY_TAGS) || \
        defined (CONFIG_CMDLINE_TAG) || \
        defined (CONFIG_INITRD_TAG) || \
        defined (CONFIG_SERIAL_TAG) || \
        defined (CONFIG_REVISION_TAG) || \
        defined (CONFIG_LCD) || \
        defined (CONFIG_VFD)
        setup_start_tag (bd);
#ifdef CONFIG_SERIAL_TAG
        setup_serial_tag (&params);
#endif
#ifdef CONFIG_REVISION_TAG
        setup_revision_tag (&params);
#endif
#ifdef CONFIG_SETUP_MEMORY_TAGS
        setup_memory_tags (bd);
#endif
#ifdef CONFIG_CMDLINE_TAG
        setup_commandline_tag (bd, commandline);
```

```
#endif
#ifdef CONFIG_INITRD_TAG
        if (initrd_start && initrd_end)
                setup_initrd_tag (bd, initrd_start, initrd_end);
#endif
#if defined (CONFIG_VFD) || defined (CONFIG_LCD)
        setup_videolfb_tag ((gd_t *) gd);
#endif
        setup_end_tag (bd);
#endif

        /* we assume that the kernel is in place */
        printf ("\nStarting kernel ...\n\n");

#ifdef CONFIG_USB_DEVICE
        {
                extern void udc_disconnect (void);
                udc_disconnect ();
        }
#endif

        /*   进入 linux 之前的清理：关中断，关 Cache 等等
                Linux 启动对 CPU 的要求：CPU 处于 SVC32 模式，中断关闭，MMU 关闭，数
                据 Cache 关闭，指令 Cache 可开可关
        */
        cleanup_before_linux ();

        /*   调用内核：R0 =0   R1= 机器类型   R2= 参数块(tagged list)地址  */
        theKernel (0, bd->bi_arch_number, bd->bi_boot_params);
}
```

5.3 linux 的内核参数传递

linux 引导的最后阶段，需要设置传递给内核的参数块，参数块的地址是物理内存起点+0x100（0x20000100 for AT91SAM9260EK）。Linux 2.6 要求使用 tagged list 的方式设置参数块。do_bootm_linux()中使用 setup_start_tag(),setup_end_tag(),setup_XXX-tag()来完成参数块的设置（参见 5.2）。具体到 AT91SAM9260EK 板，调用的函数依次是 setup_start_tag(),setup_memory_tags(),setup_commandline_tag(),setup_initrd_tag() 和 setup_end_tag()。这些函数的定义都在 lib_arm/armlinux.c 中。

(1) setup_start_tag()

[lib_arm/armlinux.c]

```
static void setup_start_tag (bd_t *bd)
{
	/*  params 指向参数块起始地址：0x20000100 for AT91SAM9260EK  */
	params = (struct tag *) bd->bi_boot_params;

	/*  设置 tag 类型：ATAG_CORE 和大小  */
	params->hdr.tag = ATAG_CORE;
	params->hdr.size = tag_size (tag_core);

	/*  设置 tag 数据  */
	params->u.core.flags = 0;
	params->u.core.pagesize = 0;
	params->u.core.rootdev = 0;

	params = tag_next (params); /*  指向下一个 tag  */

}
```

(2) setup_memory_tags()

每个 memory tag 表示一个存储区间。setup_memory_tags()设置所有的存储区间。

[lib_arm/armlinux.c]

```
static void setup_memory_tags (bd_t *bd)
{
	int i;

	for (i = 0; i < CONFIG_NR_DRAM_BANKS; i++) {
		params->hdr.tag = ATAG_MEM;	/*  设置 tag 类型 ：ATAG_MEM  */
		params->hdr.size = tag_size (tag_mem32); /*  设置 tag 大小  */

		params->u.mem.start = bd->bi_dram[i].start; /*  存储区间起点  */
		params->u.mem.size = bd->bi_dram[i].size;  /*  存储区间长度  */

		params = tag_next (params);	/*  指向下一个 tag  */
	}
}
```

(3) setup_commandline_tag()

[lib_arm/armlinux.c]

```
static void setup_commandline_tag (bd_t *bd, char *commandline)
{
	char *p;
```

```
    /*  检验参数有效性  */
    if (!commandline)
        return;

    /*  跳过空白字符  */
    for (p = commandline; *p == ' '; p++);

    /* skip non-existent command lines so the kernel will still
     * use its default command line.
     */
    if (*p == '\0')
        return;

    params->hdr.tag = ATAG_CMDLINE;  /*  设置 tag 类型 ： ATAG_CMDLINE  */

    /*  设置 tag 大小，注意大小单位是字，即 4 个字节  */
    params->hdr.size =
        (sizeof (struct tag_header) + strlen (p) + 1 + 4) >> 2;

    /*  设置 tag 数据  */
    strcpy (params->u.cmdline.cmdline, p);
    params = tag_next (params);  /*  指向下一个 tag  */
}
```

(3) setup_initrd_tag()

[lib_arm/armlinux.c]

```
static void setup_initrd_tag (bd_t *bd, ulong initrd_start, ulong initrd_end)
{
    /* an ATAG_INITRD node tells the kernel where the compressed
     * ramdisk can be found. ATAG_RDIMG is a better name, actually.
     */
    /*  设置 tag 类型 ： ATAG_INITRD2  */
    params->hdr.tag = ATAG_INITRD2;
    params->hdr.size = tag_size (tag_initrd);  /*  设置 tag 大小  */

    /*  设置 tag 数据  */
    params->u.initrd.start = initrd_start;
    params->u.initrd.size = initrd_end - initrd_start;

    /*  指向下一个 tag  */
    params = tag_next (params);
}
```

(4) setup_end_tag()

这个函数表示整个 tagged list 的结束。

[lib_arm/armlinux.c]

```
Static void setup_end_tag (bd_t *bd)
{
        params->hdr.tag = ATAG_NONE;   /*   设置 tag 类型 ： ATAG_NONE   */
        params->hdr.size = 0;   /*   设置 tag 大小 ： 0   */

}
```

至于 tag 类型的定义和基本操作，则是在 include/asm-arm/setup.h 中。

```
/*   tag_none: The list ends with an ATAG_NONE node. */
#define ATAG_NONE   0x00000000

/*   tag_ core, the list must start with an ATAG_CORE node */
#define ATAG_CORE   0x54410001
struct tag_core {
        u32 flags;       /* bit 0 = read-only */
        u32 pagesize;
        u32 rootdev;
};

/*   tag_ mem32   */
/* it is allowed to have multiple ATAG_MEM nodes */
#define ATAG_MEM     0x54410002
struct tag_mem32 {
        u32  size;
        u32  start;/* physical start address */
};

/*   tag_ videoext   */
/* VGA text type displays */
#define ATAG_VIDEOTEXT0x54410003
struct tag_videotext {
        u8          x;
        u8          y;
        u16         video_page;
        u8          video_mode;
        u8          video_cols;
        u16         video_ega_bx;
        u8          video_lines;
```

```
        u8          video_isvga;
        u16         video_points;
};

/*    tag_ ramdisk    */
/* describes how the ramdisk will be used in kernel */
#define ATAG_RAMDISK    0x54410004

struct tag_ramdisk {
        u32 flags;/* bit 0 = load, bit 1 = prompt */
        u32 size;  /* decompressed ramdisk size in _kilo_ bytes */
        u32 start; /* starting block of floppy-based RAM disk image */
};

/* describes where the compressed ramdisk image lives (virtual address) */
/*
 * this one accidentally used virtual addresses - as such,
 * its depreciated.
 */

/*    tag_ initrd    */
#define ATAG_INITRD 0x54410005
/* describes where the compressed ramdisk image lives (physical address) */
#define ATAG_INITRD2        0x54420005
struct tag_initrd {
        u32 start; /* physical start address */
        u32 size;  /* size of compressed ramdisk image in bytes */
};

/*    tag_ serialnr    */
/* board serial number. "64 bits should be enough for everybody" */
#define ATAG_SERIAL 0x54410006
struct tag_serialnr {
        u32 low;
        u32 high;
};

/*    tag_ revison    */
/* board revision */
#define ATAG_REVISION    0x54410007

struct tag_revision {
        u32 rev;
};
```

```
/*    tag_ videolfb    */
/* initial values for vesafb-type framebuffers. see struct screen_info
  * in include/linux/tty.h
  */
#define ATAG_VIDEOLFB   0x54410008

struct tag_videolfb {
        u16         lfb_width;
        u16         lfb_height;
        u16         lfb_depth;
        u16         lfb_linelength;
        u32         lfb_base;
        u32         lfb_size;
        u8          red_size;
        u8          red_pos;
        u8          green_size;
        u8          green_pos;
        u8          blue_size;
        u8          blue_pos;
        u8          rsvd_size;
        u8          rsvd_pos;
};

/*    tag_ cmdline    */
/* command line: \0 terminated string */
#define ATAG_CMDLINE    0x54410009

struct tag_cmdline {
        char cmdline[1];     /* this is the minimum size */
};

/*    tag_ acorn */
/* acorn RiscPC specific information */
#define ATAG_ACORN 0x41000101

struct tag_acorn {
        u32 memc_control_reg;
        u32 vram_pages;
        u8 sounddefault;
        u8 adfsdrives;
};

/*    tag_memclk    */
```

```
/* footbridge memory clock, see arch/arm/mach-footbridge/arch.c */
#define ATAG_MEMCLK     0x41000402

struct tag_memclk {
        u32 fmemclk;
};


/*    tag_header    &    tag    */
struct tag_header {
        u32 size;       /* size unit: words */
        u32 tag;
};

struct tag {
        struct tag_header hdr;
        union {
                struct tag_core         core;
                struct tag_mem32    mem;
                struct tag_videotext videotext;
                struct tag_ramdisk  ramdisk;
                struct tag_initrd       initrd;
                struct tag_serialnr     serialnr;
                struct tag_revision     revision;
                struct tag_videolfb     videolfb;
                struct tag_cmdline      cmdline;
                struct tag_acorn        acorn;      /*    Acorn specific    */
                struct tag_memclk   memclk;    /*    DC21285 specific    */
        } u;
};

#define tag_next(t)         ((struct tag *)((u32 *)(t) + (t)->hdr.size))
#define tag_size(type)      ((sizeof(struct tag_header) + sizeof(struct type)) >> 2)
```