U-Boot 源代码分析之三：Linux 的引导

作者：张俊岭
EMAIL: sprite_zjl@sina.com;jlzhang@tangrae.com.cn
QQ: 251450387
日期：2007-11-8
说明：本文档基于 AT91SAM9260EK 板，U-Boot 版本为 1.1.4

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。

2 引导过程

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);
}
```

3 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_VIDEOTEXT      0x54410003
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)
```