移植 U-BOOT-1.1.6 到 TQ2440

目录

一、移植相关说明

1、移植背景

天嵌公司的TQ2440 没有提供u-boot 的移植文档，移植源代码倒是有，参考U-boot1.1.6 的移植(TQ2440)，自己做了这一份移植文档，详细记录了移植步骤，修改了原文中的错误。

2、移植环境

交叉编译器：crosstools_3.4.5_softfloat
CPU：S3C2440
SDRAM：64M
Nor Flash：2M
NAND Flash：64M
网卡：DM9000

3、移植源代码

u-boot-1.1.6.tar.bz2
ftp://ftp.denx.de/pub/u-boot/

4、文件的删减

- 删除 board/下除 smdk2410 以外的所有其它目标板文件夹
- 删除 cpu/下除 arm920t 以外的所有其它 cpu 目录
- 删除根目录下 lib_XXX 的库文件目录,只留下 lib_arm 和 lib_generic
- 删除 include/目录下 asm-XXX 的文件目录,只留下 asm-arm
- 删除 include/configs 目录下除 smdk2410.h 以外的所有其它配置头文件

二、编译测试

任务：建立自己的目标板系统，并测试编译

1、解压源码包

tar -jxvf u-boot-1.1.6.tar.bz2 -C /opt/EmbedSky/
解压后在/opt/EmbedSky 目录下生成 u-boot-1.1.6 目录

2、建立自己的目标板

- 进入 u-boot-1.1.6 目录。
- ？ 将 board/smdk2410 目录复制为 dong2440 目录

cp -rf board/smdk2410/ board/dong2440

- **修改** smdk2410.c **为** dong2440.c

  mv board/dong2440/smdk2410.c board/dong2440/dong2440.c

- **？ 修改** Makefile **文件中** 28 **行的** COBJS **改为：**

```
COBJS   := dong2440.o flash.o
```

- **建立目标板配置文件：进入** include/configs **目录下，将** smdk2410.h **复制为** dong2440.h

  cp include/configs/smdk2410.h include/configs/dong2440.h

- **修改顶层（**u-boot-1.1.6 **目录）**Makefile **文件** 1881 **行，增加：**

```
dong2440_config :   unconfig
    @$(MKCONFIG) $(@:_config=) arm arm920t dong2440 NULL s3c24x0
```

各项的意思如下:

arm: CPU **的架构**(ARCH)

arm920t: CPU **的类型**(CPU),**其对应于** cpu/arm920t **子目录。**

dong2440: **开发板的型号**(BOARD),**对应于** board/dong2440 **目录。**

NULL: **开发者/或经销商**(vender)。(**此处没加** vender,**为** NULL。)

s3c24x0: **片上系统**(SOC)。

- **配置交叉编译器：修改顶层（**u-boot-1.1.6 **目录）**Makefile **文件** 128 **行，修改：**

```
ifeq ($(ARCH),arm)
CROSS_COMPILE=/opt/EmbedSky/crosstools_3.4.5_softfloat/gcc-3.4.5-glibc-2.3.6/arm-linux/bin/arm-linux-
endif
```

也可以选择自己的默认交叉编译器，此时路径不用修改。

3、**编译测试**

进入 u-boot-1.1.6 **目录**

#make mrproper //(**或**#make distclean **修改顶层** Makefile **等相关文件必须执行此步骤)**

#make dong2440_config

#make all

如果没有错误，则会生成 u-boot.bin **文件。**

至此，自己的目标板已经建立，下面要做的是修改一些配置，增加一些驱动。

三、增加对 S3C2440 的支持

任务：加入 S3C2440 **相关代码，使得** u-boot **可以在** s3c2440 **上正常工作。**

1、**修改** SDRAM **配置**

- **进入** board/dong2440 **目录修改** lowlevel_init.S **文件** 54 **行如下：**

```
#define B1_BWSCON       (DW16)          //(DW32) (IDE)
#define B2_BWSCON       (DW16)          //          (IDE)
#define B3_BWSCON       (DW16 + WAIT + UBLB)     //CS8900
#define B4_BWSCON       (DW16)                   //DM9000
#define B5_BWSCON       (DW8)           //(DW16)
#define B6_BWSCON       (DW32)
#define B7_BWSCON       (DW32)
```

- **修改** lowlevel_init.S **文件** 126 **行如下：**

```
#define REFCNT      0x4f4
                        /*period=7.8125us,HCLK=100Mhz,(2048+1-7.8125*100)=0x4F4 */
```

```
                                //1113    /* period=15.6us, HCLK=60Mhz, (2048+1-15.6*60) */
```

2、**时钟设置**

S3c2440 **的时钟计算公式和** s3c2410 **不一样，对于** s3c2440 **开发板，将** PCLK **设为** 400Mhz**，分频比为** FCLK:HCLK:PCLK=1:4:8**。**

- **首先屏蔽原来** s3c2410 **的时钟设置，修改** cpu/arm920t/**目录下** start.S **文件** 148 **行如下：**

```
#if 0
        /* FCLK:HCLK:PCLK = 1:2:4 */
        /* default FCLK is 120 MHz ! */
        ldr   r0, =CLKDIVN
        mov     r1, #3
        str   r1, [r0]
#endif
```

下面的修改可以有两种方法

方法一：

- **然后在** board_init **函数中重新配置时钟，修改** board/dong2440/dong2440.c **文件中的** 68 **行的** board_init **函数，并增加一些声明：**

```
/* S3C2440: Mpll = (2*m * Fin) / (p * 2^s), UPLL = (m * Fin) / (p * 2^s)
* m = M (the value for divider M)+ 8, p = P (the value for divider P) + 2
*/
/* Fin = 12.0000MHz */
#define S3C2440_MPLL_400MHZ     ((0x5c<<12)|(0x02<<4)|(0x01))                //HJ 400MHz
#define S3C2440_UPLL_48MHZ ((0x38<<12)|(0x02<<4)|(0x02))                     //HJ 100MHz
#define S3C2440_CLKDIV          0x05     /* FCLK:HCLK:PCLK = 1:4:8, UCLK = UPLL */
    //HJ 100MHz
/* S3C2410: Mpll,Upll = (m * Fin) / (p * 2^s)
* m = M (the value for divider M)+ 8, p = P (the value for divider P) + 2
*/
#define  S3C2410_MPLL_200MHZ        ((0x5c<<12)|(0x04<<4)|(0x00))
#define  S3C2410_UPLL_48MHZ         ((0x28<<12)|(0x01<<4)|(0x02))
#define  S3C2410_CLKDIV             0x03      /* FCLK:HCLK:PCLK = 1:2:4 */
```

以上代码针对 s3c2410、s3c2440 **分别定义了** MPLL、UPLL **寄存器的值。开发板输入时钟为** 12Mhz**（这在** include/configs/dong2440.h **中的宏** CONFIG_SYS_CLK_FREQ **中定义），读者可以根据代码中的计算公式针对自己的开发板修改系统时钟。下面是针对** s3c2410、s3c2440 **分别使用不同的宏设置系统时钟**

```
int board_init (void)
{
    S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();

    S3C24X0_GPIO * const gpio = S3C24X0_GetBase_GPIO();

    /* set up the I/O ports */
    gpio->GPACON = 0x007FFFFF;
```

```
    gpio->GPBCON = 0x00044555;
    gpio->GPBUP = 0x000007FF;
    gpio->GPCCON = 0xAAAAAAAA;
    gpio->GPCUP = 0x0000FFFF;
    gpio->GPDCON = 0xAAAAAAAA;
    gpio->GPDUP = 0x0000FFFF;
    gpio->GPECON = 0xAAAAAAAA;
    gpio->GPEUP = 0x0000FFFF;
    gpio->GPFCON = 0x000055AA;
    gpio->GPFUP = 0x000000FF;
    gpio->GPGCON = 0xFF95FFBA;
    gpio->GPGUP = 0x0000FFFF;
    gpio->GPHCON = 0x002AFAAA;
    gpio->GPHUP = 0x000007FF;
    /* support both of S3C2410 and S3C2440 */
    if ((gpio->GSTATUS1 == 0x32410000) || (gpio->GSTATUS1 == 0x32410002))
    {
        /* FCLK:HCLK:PCLK = 1:2:4 */
        clk_power->CLKDIVN = S3C2410_CLKDIV;

        /* change to asynchronous bus mod */
        __asm__(    "mrc    p15, 0, r1, c1, c0, 0\n"    /* read ctrl register    */
                    "orr    r1, r1, #0xc0000000\n"      /* Asynchronous          */
                    "mcr    p15, 0, r1, c1, c0, 0\n"    /* write ctrl register   */
                    :::"r1"
                );

        /* to reduce PLL lock time, adjust the LOCKTIME register */
        clk_power->LOCKTIME = 0xFFFFFF;

        /* configure MPLL */
        clk_power->MPLLCON = S3C2410_MPLL_200MHZ;

        /* some delay between MPLL and UPLL */
        delay (4000);

        /* configure UPLL */
        clk_power->UPLLCON = S3C2410_UPLL_48MHZ;

        /* some delay between MPLL and UPLL */
        delay (8000);

        /* arch number of SMDK2410-Board */
        gd->bd->bi_arch_number = MACH_TYPE_SMDK2410;
```

```
    }
    else
    {
        clk_power->CLKDIVN = S3C2440_CLKDIV;                    //HJ 1:4:8

        /* change to asynchronous bus mod */
        __asm__(       "mrc       p15, 0, r1, c1, c0, 0\n"       /* read ctrl register    */
                       "orr       r1, r1, #0xc0000000\n"          /* Asynchronous              */
                       "mcr     p15, 0, r1, c1, c0, 0\n"      /* write ctrl register   */
                       :::"r1"
                    );

        /* to reduce PLL lock time, adjust the LOCKTIME register */
        clk_power->LOCKTIME = 0xFFFFFF;

        /* configure MPLL */
        clk_power->MPLLCON = S3C2440_MPLL_400MHZ;        //fin=12.000MHz

        /* some delay between MPLL and UPLL */
        delay (4000);

        /* configure UPLL */
        clk_power->UPLLCON = S3C2440_UPLL_48MHZ;          //fin=12.000MHz

        /* some delay between MPLL and UPLL */
        delay (8000);

        /* arch number of SMDK2440-Board */
        gd->bd->bi_arch_number = MACH_TYPE_S3C2440;
    }
    /* adress of boot parameters */
    gd->bd->bi_boot_params = 0x30000100;

    icache_enable();
    dcache_enable();
    return 0;
}
```

最后一步，获取系统时钟的函数需要针对 s3c2410、s3c2440 **的不同进行修改。**

在后面设置串口波特率时需要获得系统时钟，就是在 U-Boot **的第二阶段**,lib_arm/board.c **中** start_armboot **函数调用** serial_init **函数初始化串口时**,**会调用** get_PCLK **函数。它在** cpu/arm920t/s3c24x0/speed.c **中定义**,**与它相关的还有** get_HCLK、get_PLLCLK **等函数。**

前面的 board_init **函数在识别出** S3C2410 **或** S3C2440 **后**,**设置了机器类型** ID:gd? bd? bi_arch_number,**后面的函数可以通过它来分辨是** S3C2410 **还是** S3C2440**。首先要在程序的开头增加如下一行**,**这样才可以使用** gd **变量。**

首先要在程序的开头增加如下一行,这样才可以使用 gd 变量。

在 cpu/arm920t/s3c24x0/speed.c 中修改：

在程序开头 40 行增加一行：DECLARE_GLOBAL_DATA_PTR;这样才可以使用 gd 变量

- 修改 get_PPLCLK 函数：

```
static ulong get_PLLCLK(int pllreg)
{
    S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
     ulong r, m, p, s;
     if (pllreg == MPLL)
    r = clk_power->MPLLCON;
     else if (pllreg == UPLL)
    r = clk_power->UPLLCON;
     else
    hang();
     m = ((r & 0xFF000) >> 12) + 8;
   p = ((r & 0x003F0) >> 4) + 2;
   s = r & 0x3;
    /* support both of S3C2410 and S3C2440 */
    if (gd->bd->bi_arch_number == MACH_TYPE_SMDK2410)
        return((CONFIG_SYS_CLK_FREQ * m) / (p << s));
     else
          return((CONFIG_SYS_CLK_FREQ * m * 2) / (p << s));    /* S3C2440 */
}
```

由于分频系数的设置方法也不一样，get_HCLK、get_PCLK 也需要修改。对于 s3c2410，沿用原来的计算方法，else 分支中是 s3c2440 的代码，如下所示：

- 修改 get_HCLK、get_PCLK：

```
/* for s3c2440 */
#define  S3C2440_CLKDIVN_PDIVN          (1<<0)
#define  S3C2440_CLKDIVN_HDIVN_MASK     (3<<1)
#define  S3C2440_CLKDIVN_HDIVN_1        (0<<1)
#define  S3C2440_CLKDIVN_HDIVN_2        (1<<1)
#define  S3C2440_CLKDIVN_HDIVN_4_8      (2<<1)
#define  S3C2440_CLKDIVN_HDIVN_3_6      (3<<1)
#define  S3C2440_CLKDIVN_UCLK           (1<<3)

#define S3C2440_CAMDIVN_CAMCLK_MASK    (0xf<<0)
#define  S3C2440_CAMDIVN_CAMCLK_SEL     (1<<4)
#define  S3C2440_CAMDIVN_HCLK3_HALF     (1<<8)
#define  S3C2440_CAMDIVN_HCLK4_HALF     (1<<9)
#define  S3C2440_CAMDIVN_DVSEN          (1<<12)
```

```
/* return HCLK frequency */
ulong get_HCLK(void)
{
```

```
    S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
    unsigned long clkdiv;
    unsigned long camdiv;
    int hdiv = 1;

    /* support both of S3C2410 and S3C2440 */
    if (gd->bd->bi_arch_number == MACH_TYPE_SMDK2410)
        return((clk_power->CLKDIVN & 0x2) ? get_FCLK()/2 : get_FCLK());
    else
    {
        clkdiv = clk_power->CLKDIVN;
        camdiv = clk_power->CAMDIVN;

        /* work out clock scalings */

        switch (clkdiv & S3C2440_CLKDIVN_HDIVN_MASK) {
        case S3C2440_CLKDIVN_HDIVN_1:
            hdiv = 1;
            break;

        case S3C2440_CLKDIVN_HDIVN_2:
            hdiv = 2;
            break;

        case S3C2440_CLKDIVN_HDIVN_4_8:
            hdiv = (camdiv & S3C2440_CAMDIVN_HCLK4_HALF) ? 8 : 4;
            break;

        case S3C2440_CLKDIVN_HDIVN_3_6:
            hdiv = (camdiv & S3C2440_CAMDIVN_HCLK3_HALF) ? 6 : 3;
            break;
        }
        return get_FCLK() / hdiv;
    }
}
```

```
/* return PCLK frequency */
ulong get_PCLK(void)
{
    S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
    unsigned long clkdiv;
    unsigned long camdiv;
    int hdiv = 1;
    /* support both of S3C2410 and S3C2440 */
```

```
    if (gd->bd->bi_arch_number == MACH_TYPE_SMDK2410)
        return((clk_power->CLKDIVN & 0x1) ? get_HCLK()/2 : get_HCLK());
    else
    {
        clkdiv = clk_power->CLKDIVN;
        camdiv = clk_power->CAMDIVN;

       /* work out clock scalings */

        switch (clkdiv & S3C2440_CLKDIVN_HDIVN_MASK) {
         case S3C2440_CLKDIVN_HDIVN_1:
              hdiv = 1;
              break;

         case S3C2440_CLKDIVN_HDIVN_2:
             hdiv = 2;
              break;

         case S3C2440_CLKDIVN_HDIVN_4_8:
            hdiv = (camdiv & S3C2440_CAMDIVN_HCLK4_HALF) ? 8 : 4;
              break;

         case S3C2440_CLKDIVN_HDIVN_3_6:
            hdiv = (camdiv & S3C2440_CAMDIVN_HCLK3_HALF) ? 6 : 3;
              break;
         }
         return get_FCLK() / hdiv / ((clkdiv & S3C2440_CLKDIVN_PDIVN)? 2:1);
    }
}
```

- **在** include/s3c24x0.h **中重新定义** S3C24X0_CLOCK_POWER **结构体，在** include/s3c24x0.h **中**,S3C24X0_CLOCK_POWER **结构体中增加**:129 **行**

```
    S3C24X0_REG32     CAMDIVN; /* for s3c2440*/
```

至此，对 s3c2440 **的支持（时钟配置部分）就算做好了，为了方便调试，可以利用开发板自带的** u-boot **文件烧写到内存中运行，此时还要修改一些配置：**

- **修改** cpu/arm920t/start.S **文件的** 162 **行如下：**

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
    @blcpu_init_crit
#endif
```

- **修改** board/dong2440/config.mk **文件如下：**

```
TEXT_BASE = 0x33000000
#TEXT_BASE = 0x33F80000
```

- **进入** u-boot-1.1.6 **目录**

```
#make distclean
#make dong2440_config
```

#make all

如果没有错误，则会生成 u-boot.bin **文件。载入内存运行出现如下信息：**

```
U-Boot 1.1.6 (Sep   4 2010 -  13:11:07)


DRAM:    64  MB
Flash:  512  kB
*** Warning - bad CRC, using default environment

In:       serial
Out:      serial
Err:      serial

SMDK2410 #
```

方法二：

此处修改时钟配置还有另一种改法，就是另外写一个时钟初始化函数，现在将其方法介绍如下。

修改方法和上面的类似，区别有以下几点：

- **修改** cpu/arm920t/start.S **文件时**

在屏蔽原来的时钟后将 stack_setup **子程序搬到** relocate **子程序之前(这步别忘了)，并在** stack_setup **子程序后加一条跳转指令调用到** clock_init **子函数，进行时钟初始化：**

增加跳转指令：

```
/* Set up the stack                                    */
stack_setup:
      ldr  r0, _TEXT_BASE          /* upper 128 KiB: relocated uboot     */
      sub  r0, r0, #CFG_MALLOC_LEN       /* malloc  area                                  */
      sub  r0, r0, #CFG_GBL_DATA_SIZE /* bdinfo                                      */
#ifdef CONFIG_USE_IRQ
      sub  r0, r0, #(CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ)
#endif
      sub  sp, r0, #12          /* leave 3 words for abort-stack       */
      bl    clock_init
#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         clear_bss
      ldr  r2, _armboot_start
      ldr  r3, _bss_start
      sub  r2, r3, r2       /* r2 <-  size  of  armboot                */
      add r2, r0, r2        /* r2 <-  source  end  address             */
```

为什么这么换呢？可能是因为所调用的 clock_init **函数需要用到堆栈。**

- **编写** clock_init **函数**

在 board/dong2440 **目录下建议一个名为** boot_init.c **的文件，编写** colck_init **函数，同时加上一些声明和延时子函数，如下：**

```
#include <common.h>
#include <s3c2410.h>
#define GSTATUS1   (*(volatile unsigned int *)0x560000B0)

static inline void delay (unsigned long loops)
{
    __asm__ volatile ("1:\n"
                "subs %0, %1, #1\n"
                "bne 1b":"=r" (loops):"0" (loops));
}

/* S3C2440: Mpll = (2*m * Fin) / (p * 2^s), UPLL = (m * Fin) / (p * 2^s)
* m = M (the value for divider M)+ 8, p = P (the value for divider P) + 2
*/
#define S3C2440_MPLL_400MHZ     ((0x5c<<12)|(0x01<<4)|(0x01))
#define  S3C2440_MPLL_200MHZ        ((0x5c<<12)|(0x01<<4)|(0x02))
#define  S3C2440_MPLL_100MHZ        ((0x5c<<12)|(0x01<<4)|(0x03))
#define  S3C2440_UPLL_96MHZ         ((0x38<<12)|(0x02<<4)|(0x01))
#define  S3C2440_UPLL_48MHZ         ((0x38<<12)|(0x02<<4)|(0x02))
#define  S3C2440_CLKDIV             (0x05) // | (1<<3))    /* FCLK:HCLK:PCLK = 1:4:8,
UCLK = UPLL/2 */
#define  S3C2440_CLKDIV188          0x04          /* FCLK:HCLK:PCLK = 1:8:8 */
#define  S3C2440_CAMDIVN188         ((0<<8)|(1<<9)) /* FCLK:HCLK:PCLK = 1:8:8 */

/* S3C2410: Mpll,Upll = (m * Fin) / (p * 2^s)
* m = M (the value for divider M)+ 8, p = P (the value for divider P) + 2
*/
#define S3C2410_MPLL_200MHZ     ((0x5c<<12)|(0x04<<4)|(0x00))
#define  S3C2410_UPLL_48MHZ         ((0x28<<12)|(0x01<<4)|(0x02))
#define S3C2410_CLKDIV              0x03          /* FCLK:HCLK:PCLK = 1:2:4 */

void clock_init(void)
{
    S3C24X0_CLOCK_POWER *clk_power = (S3C24X0_CLOCK_POWER *)0x4C000000;
        /* support both of S3C2410 and S3C2440, by www.arm9.net */
        if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
        {
            /* FCLK:HCLK:PCLK = 1:2:4 */
                clk_power->CLKDIVN = S3C2410_CLKDIV;
            /* change to asynchronous bus mod */
            __asm__("mrc    p15, 0, r1, c1, c0, 0\n" /* read ctrl register   */
                        "orr        r1, r1, #0xc0000000\n"          /* Asynchronous          */
```

```
                        "mcr     p15, 0, r1, c1, c0, 0\n" /* write ctrl register */
                        :::"r1"
                      );
            /* to reduce PLL lock time, adjust the LOCKTIME register */
            clk_power->LOCKTIME = 0xFFFFFFFF;
            /* configure UPLL */
            clk_power->UPLLCON = S3C2410_UPLL_48MHZ;
            /* some delay between MPLL and UPLL */
            delay (4000);
            /* configure MPLL */
            clk_power->MPLLCON = S3C2410_MPLL_200MHZ;
            /* some delay between MPLL and UPLL */
            delay (8000);
    }
    else
    {
            /* FCLK:HCLK:PCLK = 1:4:8 */
            clk_power->CLKDIVN = S3C2440_CLKDIV;
            /* change to asynchronous bus mod */
            __asm__("mrc    p15, 0, r1, c1, c0, 0\n" /* read ctrl register  */
                        "orr        r1, r1, #0xc0000000\n"          /* Asynchronous          */
                        "mcr     p15, 0, r1, c1, c0, 0\n" /* write ctrl register */
                        :::"r1"
                          );
            /* to reduce PLL lock time, adjust the LOCKTIME register */
            clk_power->LOCKTIME = 0xFFFFFFFF;
            /* configure UPLL */
            clk_power->UPLLCON = S3C2440_UPLL_48MHZ;
            /* some delay between MPLL and UPLL */
            delay (4000);
            /* configure MPLL */
            clk_power->MPLLCON = S3C2440_MPLL_400MHZ;
            /* some delay between MPLL and UPLL */
            delay (8000);
    }
}
```

- **然后修改** board/dong2440/dong2440.c **文件中的** board_init **函数，修改如下：**

```
int board_init (void)
{
    S3C24X0_CLOCK_POWER * const clk_power = S3C24X0_GetBase_CLOCK_POWER();
    S3C24X0_GPIO * const gpio = S3C24X0_GetBase_GPIO();

    /* set up the I/O ports */
```

```
    gpio->GPACON = 0x007FFFFF;
    gpio->GPBCON = 0x00055555;
    gpio->GPBUP = 0x000007FF;
    gpio->GPCCON = 0xAAAAAAAA;
    gpio->GPCUP = 0x0000FFFF;
    gpio->GPDCON = 0xAAAAAAAA;
    gpio->GPDUP = 0x0000FFFF;
    gpio->GPECON = 0xAAAAAAAA;
    gpio->GPEUP = 0x0000FFFF;
    gpio->GPFCON = 0x000055AA;
    gpio->GPFUP = 0x000000FF;
    gpio->GPGCON = 0xFF94FFBA;
    gpio->GPGUP = 0x0000FFEF;
    gpio->GPGDAT = gpio->GPGDAT & (~(1<<4)) | (1<<4);
    gpio->GPHCON = 0x002AFAAA;
    gpio->GPHUP = 0x000007FF;

        /*support both of S3C2410 and S3C2440*/
    if ((gpio->GSTATUS1 == 0x32410000) || (gpio->GSTATUS1 == 0x32410002))
    {
            /* arch number of SMDK2410-Board */
            gd->bd->bi_arch_number = MACH_TYPE_SMDK2410;
    }
    else
    {
           /* arch number of SMDK2440-Board */
            gd->bd->bi_arch_number = MACH_TYPE_S3C2440;
    }
    /* adress of boot parameters */
    gd->bd->bi_boot_params = 0x30000100;
    icache_enable();
    dcache_enable();
    return 0;
}
```

- **然后修改** board/dong2440/**目录下的** Makefile **文件** 28 **行，修改如下：**

```
COBJS  := dong2440.o flash.o boot_init.o
```

并在 board/dong2440/u-boot.lds **文件中** 35 **行添加如下内容：**

```
    .text      :
    {
      cpu/arm920t/start.o     (.text)
      board/dong2440/boot_init.o (.text)
      *(.text)
    }
```

以增加对 boot_init.o 的连接。
其它 include/s3c24x0.h，cpu/arm920t/s3c24x0/speed.c 的修改同上面的方法一样。
最后 make 一下，没有错误，加载到内存中运行正常。我的运行结果如下：

```
U-Boot 1.1.6 (Sep  4 2010 -  13:50:20)



DRAM:   64  MB
Flash:  512  kB
*** Warning - bad CRC, using default environment

In:       serial
Out:      serial
Err:      serial

SMDK2410  #  ?
?            -  alias  for  'help'
autoscr - run script from memory
base       -  print  or  set  address  offset
```

u-boot 中的提示符“SMDK2410 #”可以在/include/configs/dong2440.h 中修改成自己喜欢的提示符，操作如下：#define CFG_PROMPT "[dong2440]# " /* Monitor Command Prompt */

四、配置 Nor Flash

但是，现在还无法通过U-Boot命令烧写Nor Flash。本开发板中的Nor Flash型号为EN29LV160AB，而配置文件include/configs/dong2440.h中默认型号为AM29LV400。因为本开发板Nor Flash为 2MB，和AM29LV800 很相似，所以对Nor Flash配置修改如下：

```
/*-----------------------------------------------------------------------
 * FLASH and environment organization
 */
#if 0
#define CONFIG_AMD_LV400    1     /* uncomment this if you have a LV400 flash */
#endif
#define CONFIG_AMD_LV800    1     /* uncomment this if you have a LV800 flash */

#define CFG_MAX_FLASH_BANKS 1     /* max number of memory banks */
#ifdef CONFIG_AMD_LV800
#define PHYS_FLASH_SIZE         0x00200000 /* 1MB */
#define CFG_MAX_FLASH_SECT    (19)/* max number of sectors on one chip */
#define CFG_ENV_ADDR            (CFG_FLASH_BASE + 0x1F0000) /* addr of environment */
#endif
#ifdef CONFIG_AMD_LV400
```

```
#define PHYS_FLASH_SIZE		0x00080000 /* 512KB */
#define CFG_MAX_FLASH_SECT	(11) /* max number of sectors on one chip */
#define CFG_ENV_ADDR		(CFG_FLASH_BASE + 0x070000) /* addr of environment */
#endif

/* timeout values are in ticks */
#define CFG_FLASH_ERASE_TOUT	(5*CFG_HZ) /* Timeout for Flash Erase */
#define CFG_FLASH_WRITE_TOUT	(5*CFG_HZ) /* Timeout for Flash Write */

#define  CFG_ENV_IS_IN_FLASH	1
#define CFG_ENV_SIZE		0x20000		/* Total Size of Environment Sector */
```

本例中Nor Flash**的操作函数在**board/dong2440/flash.c**中实现，它支持**AM29LV400 **和**AM29LV800。

最后make**一下，没有错误，加载到内存中运行正常。我的运行结果如下：**

```
U-Boot 1.1.6 (Sep  4 2010 -  14:13:42)

DRAM:  64  MB
Flash:  2  MB
*** Warning -  bad  CRC,  using  default  environment

In:       serial
Out:      serial
Err:      serial

[dong2440]#
```

Flash: 2 MB **表示已经对**Nor Flash**支持了。**

五、增加 Nand Flash**读写驱动**

任务：移植nand-flash**驱动，让** u-boot **可以操作读写** nand flash**。由于**s3c2410 **和**s3c2440 nand flash**控制器有区别，所以修改以下代码，让**u-boot**可以操作读写**nand flash**。**

1**、增加** nand_flash.c **文件**:cpu/arm920t/s3c24x0/nand_flash.c

```
/*
 * Nand flash interface of s3c2410/s3c2440, by www.arm9.net
 * Changed from drivers/mtd/nand/s3c2410.c of kernel 2.6.13
 */
#include <common.h>
#if (CONFIG_COMMANDS & CFG_CMD_NAND) && !defined(CFG_NAND_LEGACY)
#include <s3c2410.h>
#include <nand.h>
DECLARE_GLOBAL_DATA_PTR;
#define S3C2410_NFSTAT_READY	(1<<0)
#define  S3C2410_NFCONF_nFCE		(1<<11)
#define  S3C2440_NFSTAT_READY		(1<<0)
#define  S3C2440_NFCONT_nFCE		(1<<1)
/* select chip, for s3c2410 */
```

```
static void s3c2410_nand_select_chip(struct mtd_info *mtd, int chip)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();
        if (chip == -1)
    {
            s3c2410nand->NFCONF |= S3C2410_NFCONF_nFCE;
        }
    else
    {
                s3c2410nand->NFCONF &= ~S3C2410_NFCONF_nFCE;
        }
}
/* command and control functions, for s3c2410
 *
 * Note, these all use tglx's method of changing the IO_ADDR_W field
 * to make the code simpler, and use the nand layer's code to issue the
 * command and address sequences via the proper IO ports.
 *
 */
static void s3c2410_nand_hwcontrol(struct mtd_info *mtd, int cmd)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();
        struct nand_chip *chip = mtd->priv;
        switch (cmd)
    {
            case NAND_CTL_SETNCE:
            case NAND_CTL_CLRNCE:
                    printf("%s: called for NCE\n", __FUNCTION__);
                    break;
            case NAND_CTL_SETCLE:
                    chip->IO_ADDR_W = (void *)&s3c2410nand->NFCMD;
                    break;
            case NAND_CTL_SETALE:
                    chip->IO_ADDR_W = (void *)&s3c2410nand->NFADDR;
                    break;
            default:
                    chip->IO_ADDR_W = (void *)&s3c2410nand->NFDATA;
                break;
        }
}
/* s3c2410_nand_devready()
 *
 * returns 0 if the nand is busy, 1 if it is ready
 */
```

```
static int s3c2410_nand_devready(struct mtd_info *mtd)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();
    return (s3c2410nand->NFSTAT & S3C2410_NFSTAT_READY);
}
/* select chip, for s3c2440 */
static void s3c2440_nand_select_chip(struct mtd_info *mtd, int chip)
{
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();
    if (chip == -1)
    {
        s3c2440nand->NFCONT |= S3C2440_NFCONT_nFCE;
    }
    else
    {
            s3c2440nand->NFCONT &= ~S3C2440_NFCONT_nFCE;
    }
}
/* command and control functions */
static void s3c2440_nand_hwcontrol(struct mtd_info *mtd, int cmd)
{
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();
    struct nand_chip *chip = mtd->priv;
    switch (cmd)
    {
        case NAND_CTL_SETNCE:
        case NAND_CTL_CLRNCE:
            printf("%s: called for NCE\n", __FUNCTION__);
            break;
        case NAND_CTL_SETCLE:
            chip->IO_ADDR_W = (void *)&s3c2440nand->NFCMD;
            break;
        case NAND_CTL_SETALE:
            chip->IO_ADDR_W = (void *)&s3c2440nand->NFADDR;
            break;
        default:
            chip->IO_ADDR_W = (void *)&s3c2440nand->NFDATA;
            break;
    }
}
/* s3c2440_nand_devready()
 *
 * returns 0 if the nand is busy, 1 if it is ready
 */
```

```
static int s3c2440_nand_devready(struct mtd_info *mtd)
{
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();
    return (s3c2440nand->NFSTAT & S3C2440_NFSTAT_READY);
}
/*
 * Nand flash hardware initialization:
 * Set the timing, enable NAND flash controller
 */
static void s3c24x0_nand_inithw(void)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();
    #define TACLS   0
    #define TWRPH0      4
    #define TWRPH1      2
    if (gd->bd->bi_arch_number == MACH_TYPE_SMDK2410)
    {
        /* Enable NAND flash controller, Initialize ECC, enable chip select, Set flash memory
timing */
        s3c2410nand->NFCONF =
(1<<15)|(1<<12)|(1<<11)|(TACLS<<8)|(TWRPH0<<4)|(TWRPH1<<0);
    }
    else
    {
        /* Set flash memory timing */
        s3c2440nand->NFCONF = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4);
        /* Initialize ECC, enable chip select, NAND flash controller enable */
        s3c2440nand->NFCONT = (1<<4)|(0<<1)|(1<<0);
    }
}
/*
 * Called by drivers/nand/nand.c, initialize the interface of nand flash
 */
void board_nand_init(struct nand_chip *chip)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();
    s3c24x0_nand_inithw();
    if (gd->bd->bi_arch_number == MACH_TYPE_SMDK2410)
    {
        chip->IO_ADDR_R = (void *)&s3c2410nand->NFDATA;
        chip->IO_ADDR_W = (void *)&s3c2410nand->NFDATA;
        chip->hwcontrol = s3c2410_nand_hwcontrol;
```

```
        chip->dev_ready = s3c2410_nand_devready;
        chip->select_chip = s3c2410_nand_select_chip;
        chip->options = 0;
    }
    else
    {
        chip->IO_ADDR_R = (void *)&s3c2440nand->NFDATA;
        chip->IO_ADDR_W = (void *)&s3c2440nand->NFDATA;
        chip->hwcontrol = s3c2440_nand_hwcontrol;
        chip->dev_ready = s3c2440_nand_devready;
        chip->select_chip = s3c2440_nand_select_chip;
        chip->options = 0;
    }
    chip->eccmode = NAND_ECC_SOFT;
}
#endif
```

2、**同时修改该目录下的** Makefile:29 **行**

```
COBJS  = i2c.o interrupts.o serial.o speed.o \
         usb_ohci.o  nand_flash.o
```

3、**在** include/s3c24x0.h **中定义** S3C2440_NAND **结构体**:168 **行**

```
/* NAND FLASH (see S3C2440 manual chapter 6, www.arm9.net) */
typedef struct {
    S3C24X0_REG32 NFCONF;
    S3C24X0_REG32 NFCONT;
    S3C24X0_REG32 NFCMD;
    S3C24X0_REG32 NFADDR;
    S3C24X0_REG32 NFDATA;
    S3C24X0_REG32 NFMECCD0;
    S3C24X0_REG32 NFMECCD1;
    S3C24X0_REG32 NFSECCD;
    S3C24X0_REG32 NFSTAT;
    S3C24X0_REG32 NFESTAT0;
    S3C24X0_REG32 NFESTAT1;
    S3C24X0_REG32 NFMECC0;
    S3C24X0_REG32 NFMECC1;
    S3C24X0_REG32 NFSECC;
    S3C24X0_REG32 NFSBLK;
    S3C24X0_REG32 NFEBLK;
} /*__attribute__((__packed__))*/ S3C2440_NAND;
```

4、**同时在** include/s3c2410.h **中添加**:100 **行**

```
static inline S3C2440_NAND * const S3C2440_GetBase_NAND(void)
{
    return (S3C2440_NAND * const)S3C2410_NAND_BASE;
}
```

5、**修改配置文件** include/configs/dong2440.h,**修对** Flash **的配置和增加** NAND **设置**:178 **行。**

```
//#define      CFG_ENV_IS_IN_FLASH    1
#define  CFG_ENV_IS_IN_NAND     1
#define CFG_ENV_OFFSET              0x40000
#define CFG_ENV_SIZE            0xc000/* 64M NAND FLASH Total Size of Environment Sector
*/
/*-----------------------------------------------------------------------
 * NAND flash settings
 */
#define CFG_NAND_BASE               0
#define  CFG_MAX_NAND_DEVICE            1
#define  NAND_MAX_CHIPS                 1
```

6、**修改配置文件** include/configs/dong2440.h,**增加** NAND **命令**,81 **行**

```
#define CONFIG_COMMANDS \
                (CONFIG_CMD_DFL     | \
                CFG_CMD_CACHE  | \
                CFG_CMD_NAND    | \
                /*CFG_CMD_EEPROM |*/ \
                /*CFG_CMD_I2C      |*/ \
                /*CFG_CMD_USB    |*/ \
                CFG_CMD_REGINFO   | \
                CFG_CMD_DATE    | \
                CFG_CMD_ELF)
```

最后

make clean

make all

编译成功后加载到 0x33000000 SDRAM **中运行会有** NAND **信息,输入** saveenv(**或**save) **命令后保存没有错误,输入** help **命令会多了** nand **和** nboot **命令，如下所示。**

```
U-Boot 1.1.6 (Sep  4 2010 -  14:45:26)


DRAM:    64  MB
Flash:    2  MB
NAND:    64  MiB

In:       serial
Out:      serial
Err:      serial

Hit any key to stop autoboot:  0
[dong2440]#  save
Saving Environment to NAND...
```

```
Erasing Nand...Writing to Nand... done
[dong2440]# ?
?       - alias for 'help'
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dcache  - enable or disable data cache
echo    - echo args to console
erase   - erase FLASH memory
flinfo  - print FLASH memory information
go      - start application at address 'addr'
help    - print online help
icache  - enable or disable instruction cache
iminfo  - print header information for application image
imls    - list all images found in flash
itest   - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
md      - memory display
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nand    - NAND sub-system
nboot   - boot from NAND device
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
printenv- print environment variables
protect - enable or disable FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
```

```
saveenv - save environment variables to persistent  storage
setenv   -  set  environment  variables
sleep    -  delay  execution  for  some  time
tftpboot- boot image via network using TFTP protocol
version -  print  monitor  version
[dong2440]#
```

六、支持网卡 DM9000

任务：能够用 tftp **命令下载程序到内存中运行。**

u-boot **自带网卡驱动,所以只要做些设置即可**

1、**增加网卡的** DM9000 **的配置**,include/configs/dong2440.h **的** 56 **行和** 96 **行**

```
/*
 * Hardware drivers
 */
#define CONFIG_DRIVER_DM9000  1 //去掉了原来 CS8900  的配置
#define CONFIG_DM9000_BASE      0x20000300
#define DM9000_IO             CONFIG_DM9000_BASE
#define  DM9000_DATA             (CONFIG_DM9000_BASE + 4)
#define CONFIG_DM9000_USE_16BIT

#define  CONFIG_ETHADDR          10:23:45:67:89:AB
#define  CONFIG_NETMASK          255.255.255.0
#define  CONFIG_IPADDR         10.21.17.110
#define  CONFIG_SERVERIP       10.21.17.85
```

2、driver/Makefile **里修改**:30 **行**

```
COBJS  = dm9000x.o
```

make clean

make

加载到内存中运行，如下所示。

```
[dong2440]# tftp  0x30000000  TQ2440_Test.bin
dm9000  i/o:  0x20000300,  id:  0x90000a46
MAC:  00:80:00:ff:ff:ff
operating at unknown: 15 mode
TFTP from server 10.21.17.85; our IP address is 10.21.17.110
Filename  'TQ2440_Test.bin'.
Load  address:  0x30000000
Loading: T ################################################################
             ######################
done
Bytes  transferred  =  444480  (6c840  hex)
[dong2440]#
```

出现 operating at unknown: 15 mode **错误,不过不影响使用,这是因为在网卡驱动中** drivers/dm9000x.c,**有一段程序试图连接网卡的** MII**接口,而实际上** MII **接口并未使用,所以报错,将此段程序注释掉即可。且发现网卡物理地址** MAC **不对,这是因为在显示** MAC **之前没有获取** envaddr **这个环境变量。**

1、**修改** drivers/dm9000x.c

303 **行**

将

```
 for (i = 0; i < 6; i++)
      ((u16 *) bd->bi_enetaddr)[i] = read_srom_word(i);
```

改为:

```
//start
        char *tmp = getenv("ethaddr");
        char *end;
      for  (i  =  0;  i < 6; i++)
        {
      bd->bi_enetaddr[i] = tmp ? simple_strtoul(tmp, &end, 16) : 0;
      if(tmp)
           tmp = (*end) ? end+1 : end;
        }
//end
```

331 **行**

```
#if 0
    i = 0;
    while (!(phy_read(1) & 0x20)) {    /* autonegation complete bit */
          udelay(1000);
          i++;
          if (i == 10000) {
               printf("could not establish link\n");
               return 0;
          }
    }
    /* see what we've got */
    lnk = phy_read(17) >> 12;
    printf("operating at ");
    switch (lnk) {
    case 1:
          printf("10M half duplex ");
          break;
    case 2:
          printf("10M full duplex ");
          break;
    case 4:
          printf("100M half duplex ");
          break;
    case 8:
          printf("100M full duplex ");
          break;
    default:
```

```
            printf("unknown: %d ", lnk);
            break;
        }
        printf("mode\n");
#endif
```

最后

make clean

make

加载.bin **文件到内存中并运行，显示如下：**

```
[dong2440]# tftp 0x30000000 TQ2440_Test.bin
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 10:23:45:67:89:ab
TFTP from server 10.21.17.85; our IP address is 10.21.17.110
Filename 'TQ2440_Test.bin'.
Load address: 0x30000000
Loading: T #################################################################
         ######################
done
Bytes transferred = 444480 (6c840 hex)
```

发现没有 operating at unknown: 15 mode **错误,且** MAC **显示正常。**

七、支持 NAND Flash 启动

任务：让 u-boot **支持从** nand-flash **启动(也就是同时支持** Nor Flash **和** NAND Flash **启动)(注:本阶段修改是在"时钟设置"的第二种方法修改前提进行的)。**

1、**修改** cpu/arm920t/start.S ,**修改代码搬移程序**

```
#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     clear_bss

    ldr r2, _armboot_start
    ldr r3, _bss_start
    sub r2, r3, r2      /* r2 <- size of armboot                */
#if 1
        bl  CopyCode2Ram
#else
    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
#endif    /* CONFIG_SKIP_RELOCATE_UBOOT */
```

2、**增加** CopyCode2Ram **函数及其支持子函数,修改** board/dong2440/boot_init.c **增加如下代码**

```
#include <common.h>
#include <s3c2410.h>
#define GSTATUS1   (*(volatile unsigned int *)0x560000B0)
#define  BUSY                    1
#define  NAND_SECTOR_SIZE            512
#define NAND_BLOCK_MASK              (NAND_SECTOR_SIZE - 1)
#define NAND_SECTOR_SIZE_LP        2048
#define NAND_BLOCK_MASK_LP          (NAND_SECTOR_SIZE_LP - 1)
/*  供外部调用的函数  */
void nand_init_ll(void);
void nand_read_ll(unsigned char *buf, unsigned long start_addr, int size);
/* NAND Flash  操作的总入口，它们将调用 S3C2410  或 S3C2440  的相应函数  */
static void nand_reset(void);
static void wait_idle(void);
static void nand_select_chip(void);
static void nand_deselect_chip(void);
static void write_cmd(int cmd);
static void write_addr(unsigned int addr);
static unsigned char read_data(void);
/* S3C2410  的 NAND Flash  处理函数  */
static void s3c2410_nand_reset(void);
static void s3c2410_wait_idle(void);
static void s3c2410_nand_select_chip(void);
static void s3c2410_nand_deselect_chip(void);
static void s3c2410_write_cmd(int cmd);
static void s3c2410_write_addr(unsigned int addr);
static unsigned char s3c2410_read_data(void);
/* S3C2440  的 NAND Flash  处理函数  */
static void s3c2440_nand_reset(void);
static void s3c2440_wait_idle(void);
static void s3c2440_nand_select_chip(void);
static void s3c2440_nand_deselect_chip(void);
static void s3c2440_write_cmd(int cmd);
static void s3c2440_write_addr(unsigned int addr);
static unsigned char s3c2440_read_data(void);
/* S3C2410  的 NAND Flash  操作函数  */
/*  复位  */
static void s3c2410_nand_reset(void)
{
    s3c2410_nand_select_chip();
```

```
    s3c2410_write_cmd(0xff); // 复位命令
    s3c2410_wait_idle();
    s3c2410_nand_deselect_chip();
}
/* 等待 NAND Flash 就绪 */
static void s3c2410_wait_idle(void)
{
    int i;
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2410nand->NFSTAT;
    while(!(*p & BUSY))
    for(i=0; i<10; i++);
}
/* 发出片选信号 */
static void s3c2410_nand_select_chip(void)
{
    int i;
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    s3c2410nand->NFCONF &= ~(1<<11);
    for(i=0; i<10; i++);
}
/* 取消片选信号 */
static void s3c2410_nand_deselect_chip(void)
{
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    s3c2410nand->NFCONF |= (1<<11);
}
/* 发出命令 */
static void s3c2410_write_cmd(int cmd)
{
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2410nand->NFCMD;
    *p = cmd;
}
/* 发出地址 */
static void s3c2410_write_addr(unsigned int addr)
{
    int i;
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2410nand->NFADDR;
    *p = addr & 0xff;
        for(i=0; i<10; i++);
    *p = (addr >> 9) & 0xff;
        for(i=0; i<10; i++);
```

```
    *p = (addr >> 17) & 0xff;
        for(i=0; i<10; i++);
    *p = (addr >> 25) & 0xff;
        for(i=0; i<10; i++);
}
/* 读取数据 */
static unsigned char s3c2410_read_data(void)
{
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2410nand->NFDATA;
    return *p;
}
/* S3C2440 的 NAND Flash 操作函数 */
/* 复位 */
static void s3c2440_nand_reset(void)
{
    s3c2440_nand_select_chip();
    s3c2440_write_cmd(0xff); // 复位命令
    s3c2440_wait_idle();
    s3c2440_nand_deselect_chip();
}
/* 等待 NAND Flash 就绪 */
static void s3c2440_wait_idle(void)
{
    int i;
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2440nand->NFSTAT;
    while(!(*p & BUSY))
    for(i=0; i<10; i++);
}
/* 发出片选信号 */
static void s3c2440_nand_select_chip(void)
{
    int i;
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    s3c2440nand->NFCONT &= ~(1<<1);
    for(i=0; i<10; i++);
}
/* 取消片选信号 */
static void s3c2440_nand_deselect_chip(void)
{
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    s3c2440nand->NFCONT |= (1<<1);
}
```

```
/* 发出命令 */
static void s3c2440_write_cmd(int cmd)
{
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2440nand->NFCMD;
    *p = cmd;
}
/* 发出地址 */
static void s3c2440_write_addr(unsigned int addr)
{
    int i;
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2440nand->NFADDR;
    *p = addr & 0xff;
        for(i=0; i<10; i++);
    *p = (addr >> 9) & 0xff;
        for(i=0; i<10; i++);
    *p = (addr >> 17) & 0xff;
        for(i=0; i<10; i++);
    *p = (addr >> 25) & 0xff;
        for(i=0; i<10; i++);
}
/* 发出地址 */
static void s3c2440_write_addr_lp(unsigned int addr)
{
    int i;
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2440nand->NFADDR;
    int col, page;
    col = addr & NAND_BLOCK_MASK_LP;
    page = addr / NAND_SECTOR_SIZE_LP;
    *p = col & 0xff;                    /* Column Address A0~A7 */
        for(i=0; i<10; i++);
    *p = (col >> 8) & 0x0f;             /* Column Address A8~A11 */
        for(i=0; i<10; i++);
    *p = page & 0xff;                           /* Row Address A12~A19 */
        for(i=0; i<10; i++);
    *p = (page >> 8) & 0xff;/* Row Address A20~A27 */
        for(i=0; i<10; i++);
    *p = (page >> 16) & 0x03;           /* Row Address A28~A29 */
        for(i=0; i<10; i++);
}
/* 读取数据 */
static unsigned char s3c2440_read_data(void)
```

```
{
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    volatile unsigned char *p = (volatile unsigned char *)&s3c2440nand->NFDATA;
    return *p;
}
/* 在第一次使用 NAND Flash 前,复位一下 NAND Flash */
static void nand_reset(void)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_nand_reset();
    }
    else
    {
        s3c2440_nand_reset();
    }
}
static void wait_idle(void)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_wait_idle();
    }
    else
    {
        s3c2440_wait_idle();
    }
}
static void nand_select_chip(void)
{
    int i;
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_nand_select_chip();
    }
    else
    {
        s3c2440_nand_select_chip();
    }
    for(i=0; i<10; i++);
}
```

```
static void nand_deselect_chip(void)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_nand_deselect_chip();
    }
    else
    {
        s3c2440_nand_deselect_chip();
    }
}
static void write_cmd(int cmd)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_write_cmd(cmd);
    }
    else
    {
        s3c2440_write_cmd(cmd);
    }
}
static void write_addr(unsigned int addr)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_write_addr(addr);
    }
    else
    {
        s3c2440_write_addr(addr);
    }
}
static void write_addr_lp(unsigned int addr)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        s3c2410_write_addr(addr);
    }
    else
```

```
    {
        s3c2440_write_addr_lp(addr);
    }
}
static unsigned char read_data(void)
{
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        return s3c2410_read_data();
    }
    else
    {
        return s3c2440_read_data();
    }
}
/* 初始化 NAND Flash */
void nand_init_ll(void)
{
    S3C2410_NAND * s3c2410nand = (S3C2410_NAND *)0x4e000000;
    S3C2440_NAND * s3c2440nand = (S3C2440_NAND *)0x4e000000;
    #define TACLS   0
    #define TWRPH0      3
    #define TWRPH1      0
    /* 判断是 S3C2410 还是 S3C2440 */
    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
    {
        /* 使能 NAND Flash 控制器, 初始化 ECC, 禁止片选, 设置时序 */
        s3c2410nand->NFCONF =
(1<<15)|(1<<12)|(1<<11)|(TACLS<<8)|(TWRPH0<<4)|(TWRPH1<<0);
    }
    else
    {
        /* 设置时序 */
        s3c2440nand->NFCONF = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4);
        /* 使能 NAND Flash 控制器, 初始化 ECC, 禁止片选 */
        s3c2440nand->NFCONT = (1<<4)|(1<<1)|(1<<0);
    }
    /* 复位 NAND Flash */
    nand_reset();
}
/* 读函数 */
void nand_read_ll(unsigned char *buf, unsigned long start_addr, int size)
{
```

```
    int i, j;
    if ((start_addr & NAND_BLOCK_MASK) || (size & NAND_BLOCK_MASK))
    {
        /* 地址或长度不对齐 */
        return ;
    }
    /* 选中芯片 */
    nand_select_chip();
    for(i=start_addr; i < (start_addr + size);)
    {
        /* 发出 READ0 命令 */
        write_cmd(0);
        /* Write Address */
        write_addr(i);
        wait_idle();
        for(j=0; j < NAND_SECTOR_SIZE; j++, i++)
        {
            *buf = read_data();
            buf++;
        }
    }
    /* 取消片选信号 */
    nand_deselect_chip();
    return ;
}
/* 读函数
 * Large Page
 */
void nand_read_ll_lp(unsigned char *buf, unsigned long start_addr, int size)
{
    int i, j;
    if ((start_addr & NAND_BLOCK_MASK_LP) || (size & NAND_BLOCK_MASK_LP))
    {
        /* 地址或长度不对齐 */
        return ;
    }
    /* 选中芯片 */
    nand_select_chip();
    for(i=start_addr; i < (start_addr + size);)
    {
        /* 发出 READ0 命令 */
        write_cmd(0);
        /* Write Address */
        write_addr_lp(i);
```

```
            write_cmd(0x30);
            wait_idle();
            for(j=0; j < NAND_SECTOR_SIZE_LP; j++, i++)
            {
                *buf = read_data();
                buf++;
            }
        }
        /* 取消片选信号 */
        nand_deselect_chip();
        return ;
}
int bBootFrmNORFlash(void)
{
        volatile unsigned int *pdw = (volatile unsigned int *)0;
        unsigned int dwVal;
        /*
        * 无论是从 NOR Flash 还是从 NAND Flash 启动,
        * 地址 0 处为指令"b Reset", 机器码为 0xEA00000B,
        * 对于从 NAND Flash 启动的情况,其开始 4KB 的代码会复制到 CPU 内部 4K 内存
中,
        * 对于从 NOR Flash 启动的情况,NOR Flash 的开始地址即为 0。
        * 对于 NOR Flash,必须通过一定的命令序列才能写数据,
        * 所以可以根据这点差别来分辨是从 NAND Flash 还是 NOR Flash 启动:
        * 向地址 0 写入一个数据,然后读出来,如果没有改变的话就是 NOR Flash
        */
        dwVal = *pdw;
        *pdw = 0x12345678;
        if (*pdw != 0x12345678)
        {
            return 1;
        }
        else
        {
            *pdw = dwVal;
            return 0;
        }
}
int CopyCode2Ram(unsigned long start_addr, unsigned char *buf, int size)
{
        unsigned int *pdwDest;
        unsigned int *pdwSrc;
        int i;
        if (bBootFrmNORFlash())
```

```
    {
        pdwDest = (unsigned int *)buf;
        pdwSrc = (unsigned int *)start_addr;
        /* 从 NOR Flash 启动 */
        for (i = 0; i < size / 4; i++)
        {
            pdwDest[i] = pdwSrc[i];
        }
        return 0;
    }
    else
    {
        /* 初始化 NAND Flash */
        nand_init_ll();
        /* 从 NAND Flash 启动 */
        nand_read_ll(buf, start_addr, (size +
NAND_BLOCK_MASK_LP)&~(NAND_BLOCK_MASK_LP));
        return 0;
    }
}
```

修改cpu/arm920t/start.S 162 **行**

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
    bl    cpu_init_crit
#endif
```

最后

make clean

make all

生成 u-boot.bin **文件**

然后烧到 NAND Flash **中**,**从** NAND Flash **启动**,**能够看到正常启动信息，如下所示。**

```
U-Boot 1.1.6 (Sep  4 2010 -  15:54:13)


DRAM:    64  MB
Flash:    2  MB
NAND:    64  MiB
*** Warning - bad CRC or NAND, using default environment

In:       serial
Out:      serial
Err:      serial
[dong2440]#
```

八、引导 Linux 内核

任务：让 u-boot **引导内核。**

1、**添加** u-boot **给** Linux **传递参数所需要的宏定义和环境变量配置**,

include/configs/dong2440.h

```
#include <cmd_confdefs.h>

/* for tag(s) to transfer message to kernel */
#define CONFIG_SETUP_MEMORY_TAGS                1
#define CONFIG_CMDLINE_TAG                      1
#define  CONFIG_INITRD_TAG                      1

#define CONFIG_BOOTDELAY  3
#define  CONFIG_BOOTARGS          "noinitrd root=/dev/mtdblock2 init=/linuxrc
console=ttySAC0"
#define  CONFIG_ETHADDR           10:23:45:67:89:AB
#define  CONFIG_NETMASK           255.255.255.0
#define  CONFIG_IPADDR          10.21.17.110
#define CONFIG_SERVERIP        10.21.17.85
/*#define CONFIG_BOOTFILE    "elinos-lart" */
#define CONFIG_BOOTCOMMAND "nand read 0x32000000 0x200000 0x300000; bootm
0x32000000"
```

2、**更改** mach_type **参数,修改** include/asm-arm/mach_types.h, 377 **行**

```
#define  MACH_TYPE_S3C2440                  168
```

将数值改为和内核的 mach_type **一至。至于内核的** mach_type **可以在内核** linux **源代码下的** arch/arm/tools **中的** mach_types **文件查看到。如果** mach_types **参数不匹配,会出现加载内核时到** done, booting the kernel.**停止的问题**

3、**用** mkzImage **给** zImage **加头信息**

如果将内核加载到内存中,用 bootm XXX **命令时出现** Bad Magic Number **错误时,这表明** Linux **内核缺少头信息,这就要用** mkzimage **给内核镜像加上头信息,方法如下：**

将 zImage **文件拷到** u-boot-1.1.6/tools/**目录下,输入命令：**

```
#mkimage -n 'linux-2.6.25.8' -A arm -O linux -T kernel -C none -a 0x30008000 -e 0x30008000
-d zImage zImage.img
```

则出现如下信息：

```
Image  Name:     linux-2.6.25.8
Created:         Sat  Sep   4  16:18:24  2010
Image  Type:     ARM  Linux Kernel Image (uncompressed)
Data  Size:      1728280 Bytes = 1687.77 kB = 1.65 MB
Load Address: 0x30008000
Entry Point:    0x30008000
```

这里解释一下参数的意义:

-A ==> set architecture to 'arch'
-O ==> set operating system to 'os'
-T ==> set image type to 'type'
-C ==> set compression type 'comp'
-a ==> set load address to 'addr' (hex)
-e ==> set entry point to 'ep' (hex)
-n ==> set image name to 'name'

-d ==> use image data from 'datafile'
-x ==> set XIP (execute in place)

最后

make clean

make all

生成 u-boot.bin **文件**

然后烧到 NAND Flash **中。**

```
U-Boot 1.1.6 (Sep  4 2010 -  16:21:40)


DRAM:   64  MB
Flash:   2  MB
NAND:   64  MiB

In:       serial
Out:      serial
Err:      serial

Hit any key to stop autoboot:  0
[dong2440]# tftp 0x30000000 zImage.img
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC:  10:23:45:67:89:ab
TFTP from server 10.21.17.85; our IP address is 10.21.17.110
Filename  'zImage.img'.
Load  address:  0x30000000
Loading: T ################################################################
           ################################################################
           ################################################################
           ################################################################
           ################################################################
           ##############
done
Bytes  transferred  =  1728344  (1a5f58  hex)
[dong2440]# bootm 0x30000000
##  Booting  image  at  30000000  ...
    Image  Name:     linux-2.6.25.8
    Created:          2010-09-04     8:18:24  UTC
    Image  Type:     ARM  Linux  Kernel  Image  (uncompressed)
    Data  Size:       1728280  Bytes  =   1.6  MB
    Load  Address:  30008000
    Entry  Point:   30008000
    Verifying  Checksum  ...  OK
OK
```

```
Starting kernel ...

Uncompressing Linux.....................................................
Linux version 2.6.25.8 (root@dong) (gcc version 3.4.5) #6 Thu Sep 2 20:06:48 CS0
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: SMDK2440
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C244X: core 400.000 MHz, memory 100.000 MHz, peripheral 50.000 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0
irq: clearing pending ext status 00080000
irq: clearing subpending status 00000003
irq: clearing subpending status 00000002
PID hash table entries: 256 (order: 8, 1024 bytes)
timer tcon=00500000, tcnt a2c1, tcfg 00000200,00000000, usec 00001eb8
Console: colour dummy device 80x30
console [ttySAC0] enabled
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 61312KB available (3144K code, 298K data, 124K init)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
net_namespace: 152 bytes
NET: Registered protocol family 16
S3C2410 Power Management, (c) 2004 Simtec Electronics
S3C2440: Initialising architecture
S3C2440: IRQ Support
S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
DMA channel 0 at c4800000, irq 33
DMA channel 1 at c4800040, irq 34
DMA channel 2 at c4800080, irq 35
DMA channel 3 at c48000c0, irq 36
S3C244X: Clock Support, DVS off
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
```

```
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NetWinder Floating Point Emulator V0.97 (double precision)
JFFS2 version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
fuse init (API version 7.9)
yaffs Sep  2 2010 12:02:54 Installing.
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered
io scheduler cfq registered
Console: switching to colour frame buffer device 30x40
fb0: s3c2410fb frame buffer device
lp: driver loaded but no devices found
ppdev: user-space parallel port driver
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
brd: module loaded
loop: module loaded
dm9000 Ethernet Driver
eth0: dm9000 at c485e000,c4860004 IRQ 51 MAC: 10:23:45:67:89:ab
Uniform Multi-Platform E-IDE driver
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
Driver 'sd' needs updating - please use bus_type methods
Driver 'sr' needs updating - please use bus_type methods
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c2440-nand s3c2440-nand: Tacls=2, 20ns Twrph0=3 30ns, Twrph1=2 20ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bi)
Scanning device for bad blocks
Creating 3 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00000000-0x00040000 : "TQ2440_uboot"
0x001f0000-0x003f0000 : "TQ2440_kernel"
0x003f0000-0x03ff8000 : "TQ2440_yaffs2"
usbmon: debugfs is not available
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
```

```
hub 1-0:1.0: 2 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
mice: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 390 KHz
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
drivers/hid/usbhid/hid-core.c: v2.6:USB HID core driver
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs_read_super: isCheckpointed 0
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124K
eth0: link down
/etc/rc.d/init.d/httpd: line 16: /sbin/boa: not found

Please press Enter to activate this console. eth0: link up, 100Mbps, full-duple1

[root@EmbedSky /]#
```

如上所示,u-boot**可以正常引导系统，前提是**NAND FLASH**中已经烧录好对应的文件系统。**

下面我们把内核固化在NAND FLASH**中，上电后自动引导系统。**

1.**下载内核到**SDRAM

```
[dong2440]# tftp 0x30000000 zImage.img
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 10:23:45:67:89:ab
TFTP from server 10.21.17.85; our IP address is 10.21.17.110
Filename 'zImage.img'.
Load address: 0x30000000
Loading: checksum bad
T ###################################################################
          ###################################################################
```

```
         #################################################################
         #################################################################
         #################################################################
         #############
done
Bytes transferred = 1728344 (1a5f58 hex)
```

2.**擦除**NAND FLASH，**擦除地址为** 0x200000,**大小为** 0x300000 **的**NAND FLASH

```
[dong2440]# nand erase 0x200000 0x300000

NAND erase: device 0 offset 0x200000, size 0x300000
Erasing at 0x4fc000 --  100%  complete.
OK
```

3.**写入**NAND FLASH,**将地址** 0x30000000 **的**SDRAM**的数据写入到地址** 0x200000,**大小** 0x300000 **的**NAND FLASH

```
[dong2440]# nand write 0x30000000 0x200000 0x300000

NAND write: device 0 offset 0x200000, size 0x300000
 3145728 bytes written: OK
```

4.**重新启动开发板可看到正常引导信息**

九、支持 Yaff2 文件系统

任务：u-boot- 1.1.6 **已经可以通过**"nand write"、 "nand write.jffs2'**等命令来烧写** cramfs、jffs2 **文件系统映象文件**,**下面增加**"nand write.yaffs"**命令**

1、**修改**include/configs/dong2440.h，**加入** CFG_CMD_JFFS2 **命名的定义**

```
#define CONFIG_COMMANDS \
                (CONFIG_CMD_DFL     | \
                CFG_CMD_CACHE  | \
                CFG_CMD_JFFS2    | \
                CFG_CMD_NAND    | \
                /*CFG_CMD_EEPROM |*/ \
                /*CFG_CMD_I2C      |*/ \
                /*CFG_CMD_USB    |*/ \
                CFG_CMD_REGINFO   | \
                CFG_CMD_DATE    | \
                CFG_CMD_ELF)
```

2、**增加** JFFS2 **命令行宏定义**

```
/* input clock of PLL */
#define CONFIG_SYS_CLK_FREQ    12000000/* the SMDK2410 has 12MHz input clock */

#define CONFIG_JFFS2_CMDLINE        1
#define CONFIG_JFFS2_NAND       1

#define MTDIDS_DEFAULT          "nand0=nandflash0"

#define MTDPARTS_DEFAULT        "mtdparts=nandflash0:256k@0(bios),"\
```

```
                    "48k(params)," \
                    "144k(eboot)," \
                    "1536k(logo)," \
                    "2m(kernel)," \
                    "-(root)"

#define USE_920T_MMU            1
#undef CONFIG_USE_IRQ                /* we don't need IRQ/FIQ stuff */
```

3、**在** commom/cmd_nand.c **中增加**"nand write.yaffs" **的使用说明,代码添加如下**:458 **行**

```
U_BOOT_CMD(nand, 5, 1, do_nand,
    "nand       - NAND sub-system\n",
    "info                       - show available NAND devices\n"
    "nand device [dev]      - show or set current device\n"
    "nand read[.jffs2]      - addr off|partition size\n"
    "nand write[.jffs2]     - addr off|partiton size - read/write `size' bytes starting\n"
    "     at offset `off' to/from memory address `addr'\n"
    "nand read.yaffs addr off size - read the `size' byte yaffs image starting\n"
    "     at offset `off' to memory address `addr'\n"
    "nand write.yaffs addr off size - write the `size' byte yaffs image starting\n"
    "     at offset `off' from memory address `addr'\n"
    "nand erase [clean] [off size] - erase `size' bytes from\n"
    "      offset `off' (entire device if not specified)\n"
    "nand bad - show bad blocks\n"
    "nand dump[.oob] off - dump page\n"
    "nand scrub - really clean NAND erasing bad blocks (UNSAFE)\n"
    "nand markbad off - mark bad block at offset (UNSAFE)\n"
    "nand biterr off - make a bit error at offset (UNSAFE)\n"
    "nand lock [tight] [status] - bring nand to lock state or display locked pages\n"
    "nand unlock [offset] [size] - unlock section\n");
```

4、**在** nand **命令的处理函数** do_nand **中增加对**"nand yaffs"**的支持。**do_nand **函数仍在** commom/cmd_nand.c **中实现,代码修改如下**: 354 **行**

```
if (s != NULL &&
        (!strcmp(s, ".jffs2") || !strcmp(s, ".e") || !strcmp(s, ".i"))) {
            if (read) {
                    /* read */
                     nand_read_options_t opts;
                     memset(&opts, 0, sizeof(opts));
                     opts.buffer          = (u_char*) addr;
                     opts.length          = size;
                     opts.offset          = off;
                     opts.quiet               = quiet;
                     ret = nand_read_opts(nand, &opts);
            } else {
                    /* write */
```

```
                        nand_write_options_t  opts;
                         memset(&opts, 0, sizeof(opts));
                        opts.buffer          =  (u_char*) addr;
                        opts.length          =  size;
                        opts.offset          =  off;
                       /*  opts.forcejffs2  =  1;  */
                        opts.pad             =  1;
                       opts.blockalign  =  1;
                        opts.quiet             =  quiet;
                        ret  =  nand_write_opts(nand, &opts);
                }
}else if ( s != NULL && !strcmp(s, ".yaffs")){
        if  (read)  {
                /*  read  */
                 nand_read_options_t  opts;
                  memset(&opts, 0, sizeof(opts));
                 opts.buffer  =  (u_char*) addr;
                opts.length  =  size;
                opts.offset  =  off;
                opts.readoob  =  1;
                 opts.quiet             =  quiet;
                ret  =  nand_read_opts(nand, &opts);
        }  else  {
                 /* write */
                 nand_write_options_t opts;
                 memset(&opts, 0, sizeof(opts));
                opts.buffer  =  (u_char*)  addr;
                opts.length  =  size;
                opts.offset  =  off;
                /*  opts.forceyaffs = 1; */
                opts.noecc  = 1;
                opts.writeoob  =  1;

                opts.blockalign  =  1;
                 opts.quiet             =  quiet;
                opts.skipfirstblk  =  1;
                ret  = nand_write_opts(nand, &opts);
         }
}
        else  {
                if (read)
                        ret = nand_read(nand, off, &size, (u_char *)addr);
                else
                        ret  =  nand_write(nand, off, &size, (u_char *)addr);
```

```
        }
        printf(" %d bytes %s: %s\n", size,
            read ? "read" : "written", ret ? "ERROR" : "OK");
return ret == 0 ? 0 : 1;
```

上述代码中,opts.skipfirstblk **是新增加的项,表示烧写时跳过第一个可用的逻辑块，这是由**yaffs**文件系统的特性决定的。下面给**opts.skipfirstblk **新增加项重新定义** nand_write_options_t **结构,并在下面调用的** nand_write_opts **函数中对他进行处理。**

5、**在** include/nand.h **中进行如下修改,增加** skipfirstblk **成员**: 81 **行**

```
struct nand_write_options {
    u_char *buffer;         /* memory block containing image to write */
    ulong length;           /* number of bytes to write */
    ulong offset;       /* start address in NAND */
    int quiet;          /* don't display progress messages */
    int autoplace;          /* if true use auto oob layout */
    int forcejffs2;         /* force jffs2 oob layout */
    int forceyaffs;         /* force yaffs oob layout */
    int noecc;          /* write without ecc */
    int writeoob;       /* image contains oob data */
    int pad;        /* pad to page size */
    int blockalign;         /* 1|2|4 set multiple of eraseblocks
                     * to align to */
    int skipfirstblk;
};
```

6、**在** drivers/nand/nand_util.c **修改** nand_write_opts **函数,增加对** skipfirstblk **成员的支持**

301 **行**

```
int nand_write_opts(nand_info_t *meminfo, const nand_write_options_t *opts)
{
    int imglen = 0;
    int pagelen;
    int baderaseblock;
    int blockstart = -1;
    loff_t offs;
    int readlen;
    int oobinfochanged = 0;
    int percent_complete = -1;
    struct nand_oobinfo old_oobinfo;
    ulong mtdoffset = opts->offset;
    ulong erasesize_blockalign;
    u_char *buffer = opts->buffer;
    size_t written;
    int result;
    int skipfirstblk = opts->skipfirstblk;
    if (opts->pad && opts->writeoob) {
```

```
            printf("Can't pad when oob data is present.\n");
            return -1;
        }
```

428 **行**

```
/* skip the first good block when wirte yaffs image, by www.embedsky.net */
            if (skipfirstblk) {
                mtdoffset += erasesize_blockalign;
                    skipfirstblk = 0;
                    continue;
            }
            readlen = meminfo->oobblock;
            if (opts->pad && (imglen < readlen)) {
                readlen = imglen;
                memset(data_buf + readlen, 0xff,
                            meminfo->oobblock - readlen);
            }
```

进行上面移植后,u-boot **已经支持** yaffs **文件系统映象的烧写**,**由于前面设**"opts.noecc=1" **不使用** ECC **校验码**,**烧写时会提示很多提示信息。**

7、**修改** drivers/nand/nand_base.c **文件中的** nand_write_page **函数**, **将其注释掉。**

910 **行**

```
case NAND_ECC_NONE:
            //printk (KERN_WARNING "Writing data without ECC to NAND-FLASH is not
recommended\n");
            this->write_buf(mtd, this->data_poi, mtd->oobblock);
            break;
```

8、**完善** do_go **函数，编辑** common/cmd_boot.c **函数即可。**

```
#include <common.h>
#include <command.h>
#include <net.h>

#if defined(CONFIG_I386)
DECLARE_GLOBAL_DATA_PTR;
#endif

void call_linux(long a0, long a1, long a2)
{
    __asm__(" mov r1, #0\n"
            " mov r1, #7 << 5\n" /* 8 segments */
            "1: orr r3, r1, #63 << 26\n" /* 64 entries */
            "2: mcr p15, 0, r3, c7, c14, 2\n" /* clean & invalidate D index */
            " subs r3, r3, #1 << 26\n"
            " bcs 2b\n" /* entries 64 to 0 */
            " subs r1, r1, #1 << 5\n"
            " bcs 1b\n" /* segments 7 to 0 */
```

```
            " mcr p15, 0, r1, c7, c5, 0\n" /* invalidate I cache */
            " mcr p15, 0, r1, c7, c10, 4\n" /* drain WB */
            );
    __asm__("mov r0, #0\n"
            "mcr p15, 0, r0, c7, c10, 4\n" /* drain WB */
            "mcr p15, 0, r0, c8, c7, 0\n" /* invalidate I & D TLBs */
            );
    __asm__(
            "mov r0, %0\n"
            "mov r1, #0x0c1\n"
            "mov r2, %2\n"
            "mov ip, #0\n"
            "mcr p15, 0, ip, c13, c0, 0\n" /* zero PID */
            "mcr p15, 0, ip, c7, c7, 0\n" /* invalidate I,D caches */
            "mcr p15, 0, ip, c7, c10, 4\n" /* drain write buffer */
            "mcr p15, 0, ip, c8, c7, 0\n" /* invalidate I,D TLBs */
            "mrc p15, 0, ip, c1, c0, 0\n" /* get control register */
            "bic ip, ip, #0x0001\n" /* disable MMU */
            "mcr p15, 0, ip, c1, c0, 0\n" /* write control register */
            "mov pc, r2\n"
            "nop\n"
            "nop\n"
            : /* no outpus */
            : "r" (a0), "r" (a1), "r" (a2)
            );
}

static void setup_linux_param(ulong param_base)
{
    struct param_struct *params = (struct param_struct *)param_base;
    char *linux_cmd;
    //linux_cmd = "noinitrd root=/dev/mtdblock/2 init=/linuxrc console=ttyS0";
    linux_cmd = getenv("bootargs");
    memset(params, 0, sizeof(struct param_struct));
    params->u1.s.page_size = 0x00001000;
    params->u1.s.nr_pages = (0x04000000 >> 12);
    /* set linux command line */
    memcpy(params->commandline, linux_cmd, strlen(linux_cmd) + 1);
}

int do_go (cmd_tbl_t *cmdtp, int flag, int argc, char *argv[])
{
    ulong    addr, rc;
    int      rcode  =  0;
```

```
    if (argc < 2) {
        printf ("Usage:\n%s\n", cmdtp->usage);
        return 1;
    }

    addr = simple_strtoul(argv[1], NULL, 16);
    printf ("## Starting application at 0x%08lX ...\n", addr);
    setup_linux_param(0x30000100);
    call_linux(0,0x0c1,0x30008000);
    printf("ok\n");
    printf ("## Starting application at 0x%08lX ...\n", addr);
    /*
     * pass address parameter as argv[0] (aka command name),
     * and all remaining args
     */
#if defined(CONFIG_I386)
    /*
     * x86 does not use a dedicated register to pass the pointer
     * to the global_data
     */
    argv[0] = (char *)gd;
#endif
#if !defined(CONFIG_NIOS)
    rc = ((ulong (*)(int, char *[]))addr) (--argc, &argv[1]);
#else
    /*
     * Nios function pointers are address >> 1
     */
    rc = ((ulong (*)(int, char *[]))(addr>>1)) (--argc, &argv[1]);
#endif
    if (rc != 0) rcode = 1;

    printf ("## Application terminated, rc = 0x%lX\n", rc);
    return rcode;
}
```

最后

make

没有错误。至此,Uboot **的移植就已经移好了。**

十、烧写 Yaff2 文件系统

1.**下载文件系统到**SDRAM

```
[dong2440]# tftp 0x30000000 root_2.6.25.8.yaffs
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 10:23:45:67:89:ab
```

```
TFTP from server 10.21.17.85; our IP address is 10.21.17.110
Filename  'root_2.6.25.8.yaffs'.
Load  address:  0x30000000
Loading: T #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
           #################################################################
```

```
         #################################################################
         #################################################################
         #################################################################
         ###########################################################
done
Bytes transferred = 14923920 (e3b890 hex)
```

2.**擦除**NAND FLASH，**擦除地址为** 0x500000,**大小为** 0x3b00000 **的**NAND FLASH

```
[dong2440]# nand erase 0x500000 0x3b00000

NAND erase: device 0 offset 0x500000, size 0x3b00000
Erasing at 0x3ffc000 --  100% complete.
OK
```

3.**写入**NAND FLASH,**将地址** 0x30000000 **的**SDRAM**的数据写入到地址** 0x500000,**大小** 0xe3b890 **的**NAND FLASH

```
[dong2440]# nand write.yaffs 0x30000000 0x500000 0xe3b890

NAND write: device 0 offset 0x500000, size 0xe3b890

Writing data at 0x12d1000 --  100% complete.
 14923920 bytes written: OK
```

4.**重新启动板子，内核启动后出现如下信息**

```
page 2251 in gc has no object: 0 0 0
page 2252 in gc has no object: 0 0 0
page 2253 in gc has no object: 0 0 0
page 2254 in gc has no object: 0 0 0
page 2255 in gc has no object: 0 0 0
page 2256 in gc has no object: 0 0 0
page 2257 in gc has no object: 0 0 0
page 2258 in gc has no object: 0 0 0
page 2259 in gc has no object: 0 0 0
```

在u-boot**命令模式下输入，即可解决**

```
[dong2440]# mtdparts default
[dong2440]# save
Saving Environment to NAND...
Erasing Nand...Writing to Nand... done
```