本文档采用了 Uboot1.1.6 中的 nandflash 的新驱动，没有用 nand_legacy，同时添加了 yaffs 文件系统烧写的功能，并且对网上一些移植文档的不妥，缺少之处进行补充。如有不妥之处，欢迎指正。

联系方式：edaworld@yeah.net

零、移植前说明：

1. 工作环境：

Fedora 7 ,内核 2.6.21

交叉编译器：

Arm-linux-gcc 3.3.2

目标板：

优龙 FS2410，NAND Flash:64M K9F1208，NOR Flash:2M SST39VF1601 (本次移植不包含 NOR Flash 支持)， RAM 64M ， CS8900Q3

2. 下载源码，建立工作目录

u-boot 的源码可以从以下网址下载：

http://downloads.sourceforge.net/u-boot/u-boot-1.1.6.tar.bz2

建立工作目录：

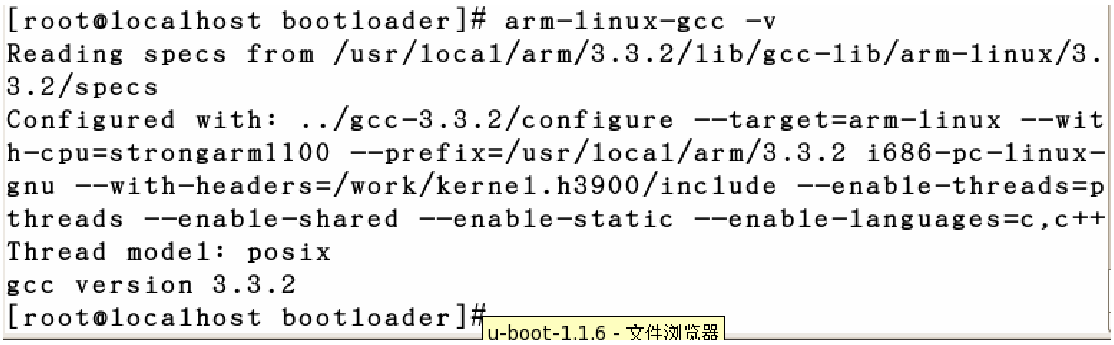
mkdir /bootloader

cd /bootloader

把下载的源码拷贝到该目录，解压；

tar jxvf u-boot-1.1.6.tar.bz2

注意使用交叉编译器为 3.3.2 版本



```
[root@localhost bootloader]# arm-linux-gcc -v
Reading specs from /usr/local/arm/3.3.2/lib/gcc-lib/arm-linux/3.3.2/specs
Configured with: ../gcc-3.3.2/configure --target=arm-linux --with-cpu=strongarm1100 --prefix=/usr/local/arm/3.3.2 i686-pc-linux-gnu --with-headers=/work/kernel.h3900/include --enable-threads=pthreads --enable-shared --enable-static --enable-languages=c,c++
Thread model: posix
gcc version 3.3.2
[root@localhost bootloader]#
```

一、移植步骤如下：

（1）、建立自己 fs2410 开发板的配置

cd /u-boot-1.1.6

1)# cp –r board/smdk2410 board/fs2410

2)# cp include/configs/smdk2410.h include/configs/fs2410.h

fs2410.h 是开发板的配置文件，他包括开发板的 CPU、系统时钟、RAM、FLASH 系统及其他相关的配置信息，由于 u-boot 已经支持三星的 SMDK2410 开发板，所以移植的时候直接拷贝 SMDK2410 的配置文件，做相应的修改即可。由于 Uboot 对 SMDK2410 板的 NAND Flash 初始化部分没有写，即 lib_arm/board.c 中的 start_armboot 函数中有这

么一句：

```
#if (CONFIG_COMMANDS & CFG_CMD_NAND)
puts ("NAND:");
nand_init(); /* go init the NAND */
#endif
```

但是在 board/smdk2410 目录下源文件中都没有定义 nand_init 这个函数。所以需要我们补充这个函数以及这个函数涉及的底层操作，NAND Flash 的读写操作相对复杂，将在 u-boot-1.1.6 移植的后面部分介绍。

(2). 修改顶层 Makefile

```
cd /u-boot-1.1.6
gedit Makefile
```

找到：

```
smdk2410_config     :     unconfig
    @$(MKCONFIG) $(@:_config=) arm arm920t smdk2410 NULL s3c24x0
```

在其后面添加：

```
fs2410_config     :     unconfig
    @$(MKCONFIG) $(@:_config=) arm arm920t fs2410 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。
fs2410:   开发板的型号(BOARD)，对应于 board/fs2410 目录。
NULL: 开发者/或经销商(vender)。
s3c24x0:   片上系统(SOC)。

(3). 修改 include/configs/fs2410.h:

修改：

```
# define    CFG_PROMPT      “SMDK2410 #”
```

为：

```
# define    CFG_PROMPT      “[Neusoft2410]#”
```

这是 u-boot 的命令行提示符。

此处是 bootloader 启动后的提示符定义。

(4) 修改 board/fs2410/Makefile

将：

```
COBJS     := smdk2410.o flash.o
```

改为：

```
COBJS     := fs2410.o flash.o
```

当然，fs2410 下的 smdk2410.c 要改成 fs2410.c;

(5）依照你自己开发板的内存地址分配情况修改 board/fs2410/lowlevel_init.S 文件

这里我参考了FS2410开发板自带S3C2410_BIOS,代码如下：

```
#include <config.h>
#include <version.h>


/* some parameters for the board */

/*
 *
 * Taken from linux/arch/arm/boot/compressed/head-s3c2410.S
 *
 * Copyright (C) 2002 Samsung Electronics
SW.LEE  <hitchcar@sec.samsung.com>
 *
 */

#define BWSCON    0x48000000

/* BWSCON */
#define DW8             (0x0)
#define DW16             (0x1)
#define DW32             (0x2)
#define WAIT            (0x1<<2)
#define UBLB            (0x1<<3)

#define B1_BWSCON          (DW16)
#define B2_BWSCON          (DW16)
#define B3_BWSCON          (DW16 + WAIT + UBLB)
#define B4_BWSCON          (DW16)
#define B5_BWSCON          (DW16)
#define B6_BWSCON          (DW32)
#define B7_BWSCON          (DW32)

/* BANK0CON */
#define B0_Tacs           0x3   /*  0clk */
#define B0_Tcos           0x3   /*  0clk */
#define B0_Tacc           0x7   /* 14clk */
#define B0_Tcoh           0x3   /*  0clk */
#define B0_Tah           0x3   /*  0clk */
#define B0_Tacp           0x1
#define B0_PMC           0x0   /* normal */

/* BANK1CON */
#define B1_Tacs           0x3   /*  0clk */
```

```
#define B1_Tcos          0x3    /*  0clk */
#define B1_Tacc          0x7    /* 14clk */
#define B1_Tcoh          0x3    /*  0clk */
#define B1_Tah           0x3    /*  0clk */
#define B1_Tacp          0x3
#define B1_PMC           0x0

#define B2_Tacs          0x0
#define B2_Tcos          0x0
#define B2_Tacc          0x7
#define B2_Tcoh          0x0
#define B2_Tah           0x0
#define B2_Tacp          0x0
#define B2_PMC           0x0

#define B3_Tacs          0x0    /*  0clk */
#define B3_Tcos          0x3    /*  4clk */
#define B3_Tacc          0x7    /* 14clk */
#define B3_Tcoh          0x1    /*  1clk */
#define B3_Tah           0x0    /*  0clk */
#define B3_Tacp          0x3     /*  6clk */
#define B3_PMC           0x0    /* normal */

#define B4_Tacs          0x1    /*  0clk */
#define B4_Tcos          0x1    /*  0clk */
#define B4_Tacc          0x6    /* 14clk */
#define B4_Tcoh          0x1    /*  0clk */
#define B4_Tah           0x1    /*  0clk */
#define B4_Tacp          0x0
#define B4_PMC           0x0    /* normal */

#define B5_Tacs          0x1    /*  0clk */
#define B5_Tcos          0x1    /*  0clk */
#define B5_Tacc          0x6    /* 14clk */
#define B5_Tcoh          0x1    /*  0clk */
#define B5_Tah           0x1    /*  0clk */
#define B5_Tacp          0x0
#define B5_PMC           0x0    /* normal */

#define B6_MT           0x3    /* SDRAM */
#define B6_Trcd           0x1
#define B6_SCAN           0x1    /* 9bit */

#define B7_MT           0x3    /* SDRAM */
```

```
#define B7_Trcd             0x1    /* 3clk */
#define B7_SCAN             0x1    /* 9bit */

/* REFRESH parameter */
#define REFEN              0x1    /* Refresh enable */
#define TREFMD             0x0    /* CBR(CAS before RAS)/Auto refresh */
#define Trp              0x0    /* 2clk */
#define Trc              0x3    /* 7clk */
#define Tchr              0x2    /* 3clk */
#define REFCNT             1113    /* period=15.6us, HCLK=60Mhz,
(2048+1-15.6*60) */
/**************************************/

_TEXT_BASE:
    .word    TEXT_BASE

.globl lowlevel_init
lowlevel_init:
    /* memory control configuration */
    /* make r0 relative the current location so that it */
    /* reads SMRDATA out of FLASH rather than memory ! */
    ldr     r0, =SMRDATA
    ldr    r1, _TEXT_BASE
    sub    r0, r0, r1
    ldr    r1, =BWSCON    /* Bus Width Status Controller */
    add     r2, r0, #13*4
0:
    ldr     r3, [r0], #4
    str     r3, [r1], #4
    cmp     r2, r0
    bne     0b

    /* everything is fine now */
    mov    pc, lr

    .ltorg
/* the literal pools origin */

SMRDATA:
    .word
(0+(B1_BWSCON<<4)+(B2_BWSCON<<8)+(B3_BWSCON<<12)+(B4_BWS
CON<<16)+(B5_BWSCON<<20)+(B6_BWSCON<<24)+(B7_BWSCON<<28))
    .word
((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah
<<4)+(B0_Tacp<<2)+(B0_PMC))
    .word
((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah
<<4)+(B1_Tacp<<2)+(B1_PMC))
    .word
((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah
<<4)+(B2_Tacp<<2)+(B2_PMC))
    .word
((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah
<<4)+(B3_Tacp<<2)+(B3_PMC))
    .word
((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah
<<4)+(B4_Tacp<<2)+(B4_PMC))
    .word
((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah
<<4)+(B5_Tacp<<2)+(B5_PMC))
    .word ((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
    .word ((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
    .word
((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFC
NT)
    .word 0x32
    .word 0x30
    .word 0x30
```

（6）测试编译能否成功：

执行

```
    make fs2410_config
    make
```

如果没有问题，在 u-boot-1.1.6 目录下就生成 u-boot.bin，因为到这一步只是做了点小改动，并未涉及敏感问题，测试一下可增加点信心，烧到扳子看到如图 1 所示。当然也有 make 不成功的时候，如按照上述步骤编译 u-boot-1.1.5 的时候，出现“没有规则创建'all'需要的目标'hello_world.srec'”,如图 1 所示，解决方法：

把 example 文件夹下的 Makefile 中的

第 147 行

%.srec: % 改成： %.srec: %.o

第 150 行

%.bin: % 改成： %.bin: %.o

网上还有一种改法，我没试过，不作说明。

（7）在 board/fs2410 加入 NAND Flash 读函数，建立 nand_read.c，加入如下内容(copy from vivi)：

```
#include <config.h>
```

```
#include "linux/mtd/mtd.h"
#include "linux/mtd/nand.h"

#define __REGb(x) (*(volatile unsigned char *)(x))
#define __REGi(x) (*(volatile unsigned int *)(x))
#define NF_BASE 0x4e000000
#define NFCONF __REGi(NF_BASE + 0x0)
#define NFCMD __REGb(NF_BASE + 0x4)
#define NFADDR __REGb(NF_BASE + 0x8)
#define NFDATA __REGb(NF_BASE + 0xc)
#define NFSTAT __REGb(NF_BASE + 0x10)
#define BUSY 1
inline void wait_idle(void) {
        int i;
        while(!(NFSTAT & BUSY))
            for(i=0; i<10; i++);
}
#define NAND_SECTOR_SIZE 512
#define NAND_BLOCK_MASK (NAND_SECTOR_SIZE - 1)
/* low level nand read function */
int
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)) {
                return -1; /* invalid alignment */
        }
        /* chip Enable */
        NFCONF &= ~0x800;
        for(i=0; i<10; i++);
          for(i=start_addr; i < (start_addr + size);) {
            /* READ0 */
            NFCMD = 0;
              /* Write Address */
            NFADDR = i & 0xff;
            NFADDR = (i >> 9) & 0xff;
            NFADDR = (i >> 17) & 0xff;
            NFADDR = (i >> 25) & 0xff;
              wait_idle();
              for(j=0; j < NAND_SECTOR_SIZE; j++, i++) {
*buf = (NFDATA & 0xff);
buf++;
            }
        }
```

```
     /* chip Disable */
    NFCONF |= 0x800; /* chip disable */
    return 0;
}
```

（8）接着修改 board/fs2410/Makefile

```
    COBJS := fs2410.o flash.o nand_read.o
```

(9）修改 cpu/arm920t/start.S 文件

2410 的启动代码可以在外部的 NAND FLASH 上执行，启动时，NAND FLASH 的前 4KB（地址为 0x00000000，OM[1:0]=0）将被装载到 SDRAM 中被称为 Setppingstone 的地址中，然后开始执行这段代码。启动以后，这 4KB 的空间可以做其他用途,在 start.S 加入搬运代码如下：

```
..........
..........
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
```

下面红色是要添加的内容，添加到蓝色的部分中间，蓝色是已经有的代码

```
/*********************************************************/
#ifdef CONFIG_S3C2410_NAND_BOOT /*这个一定要放在堆栈设置之前*/
    bl      copy_myself
#endif      /*CONFIG_S3C2410_NAND_BOOT*/
/*********************************************************/

#endif      /* CONFIG_SKIP_RELOCATE_UBOOT */

    /* Set up the stack                                    */
stack_setup:
.................
…………..
/*************************************************************************
 *
 * copy u-boot   to ram  放在 start.S 靠后的位置
 *
 *************************************************************************
 */
 #ifdef CONFIG_S3C2410_NAND_BOOT
/*
@ copy_myself: copy u-boot to ram
*/
copy_myself:
    mov     r10, lr

    @ reset NAND
    mov      r1, #NAND_CTL_BASE
    ldr     r2, =0xf830             @ initial value
    str     r2, [r1, #oNFCONF]
    ldr     r2, [r1, #oNFCONF]
    bic     r2, r2, #0x800          @ enable chip
    str     r2, [r1, #oNFCONF]
    mov      r2, #0xff              @ RESET command
    strb     r2, [r1, #oNFCMD]
    mov      r3, #0                 @ wait
1:      add      r3, r3, #0x1
    cmp      r3, #0xa
    blt     1b
2:      ldr      r2, [r1, #oNFSTAT]     @ wait ready
    tst     r2, #0x1
    beq      2b
    ldr     r2, [r1, #oNFCONF]
    orr     r2, r2, #0x800          @ disable chip
    str     r2, [r1, #oNFCONF]

    @ get read to call C functions
    ldr     sp, DW_STACK_START      @ setup stack pointer
    mov      fp, #0                 @ no previous frame, so fp=0

    @ copy UBOOT to RAM
    ldr     r0, _TEXT_BASE
    mov        r1, #0x0
    mov      r2, #0x40000
    bl      nand_read_ll

    teq     r0, #0x0
    beq      ok_nand_read

bad_nand_read:
1:      b       1b              @ infinite loop

ok_nand_read:

    @ verify

    mov      r0, #0
    ldr     r1, _TEXT_BASE
    mov       r2, #0x400        @ 4 bytes * 1024 = 4K-bytes
go_next:
    ldr       r3, [r0], #4
    ldr       r4, [r1], #4
    teq       r3, r4
    bne       notmatch
    subs      r2, r2, #4
    beq       done_nand_read
    bne       go_next
notmatch:
1:      b       1b
done_nand_read:

    mov      pc, r10

#endif
    @ CONFIG_S3C2440_NAND_BOOT

DW_STACK_START:
    .word       STACK_BASE+STACK_SIZE-4
```

(10）修改 include/configs/fs2410.h 文件，添加如下内容：

```
/****************************************************/
/*---------------------------------------------------------------------
  *   NAND FLASH BOOT
  */
#define      CONFIG_S3C2410_NAND_BOOT       1
#define      STACK_BASE                     0x33f00000
#define      STACK_SIZE                         0x8000
#define      UBOOT_RAM_BASE                      0x30100000

#define      NAND_CTL_BASE                       0x4e000000
#define      bINT_CTL(Nb)                   _REG(INT_CTL_BASE+(Nb))


#define      oNFCONF                        0x00
#define      oNFCMD                        0x04
#define      oNFADDR                        0x08
#define      oNFDATA                        0x0c
#define      oNFSTAT                       0x10
#define      oNFECC                       0x14
/*-------------------------------------------------------------------*/
#define NAND_MAX_CHIPS              1
```

(11）重新编译 u-boot

```
make    fs2410_config
make
```

(12)通过 fs2410 的 NOR FLASH 上的 BIOS 将 u-boot.bin 烧写到 nand flash 中就可以从 NAND flash 启动了

我的 u-boot 启动信息如下所示，

```
U-Boot 1.1.6 (May 19 2009 - 14:37:43)

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

In:       serial
Out:      serial
Err:      serial
FS2410 #
```

可以看出：和第一次 make 的结果一样，u-boot 命令依然不能用，也就是说不能用 saveenv 保存设置，因为我们现在只是完成了 u-boot 从 NAND FLASH 的启动工作，添加了 nand_read.c 函数，而不能实现写操作，下面将实现 u-boot 的一些命令，tftp、 saveenv、 go 等。

二、修改配置文件 include/configs/s3c2410.h 使支持 NAND 及添加修改一些参数的设置

(1)修改配置文件

```
#define CONFIG_COMMANDS \
            (CONFIG_CMD_DFL      | \
            CFG_CMD_CACHE      | \
            CFG_CMD_ENV        | \
            CFG_CMD_NET        | \
            CFG_CMD_PING     | \
            CFG_CMD_NAND     | \
            CFG_CMD_REGINFO  | \
            CFG_CMD_DATE     | \
            CFG_CMD_ELF)

#define CFG_NAND_BASE         0x4E000000
#define CFG_MAX_NAND_DEVICE     1
#define NAND_MAX_CHIPS          1


#define CFG_ENV_IS_IN_NAND     1
#define CMD_SAVEENV
#define CFG_ENV_SIZE              0x10000
#define CFG_ENV_OFFSET        0x30000
```

（2）下面的内容可以根据自己的需要进行改动

```
#define CONFIG_BOOTDELAY    3
#define CONFIG_BOOTARGS        "noinitrd root=/dev/mtdblock2
init=/linuxrc devfs=mount console=ttySAC0,115200"
#define CONFIG_ETHADDR    08:00:3e:26:0a:5b
#define CONFIG_NETMASK          255.255.255.0
#define CONFIG_IPADDR        202.193.74.101
#define CONFIG_SERVERIP        202.193.74.235
#define CONFIG_BOOTCOMMAND    "nand read 0x30008000 0x40000 0x1c0000;
bootm 0x30008000"
```

三、建立 cpu/arm920t/s3c24x0/nand_flash.c，实现 board_nand_init 函数，能够实现 nand 的写操作。同时包含对 S3C2440 的支持，一并列出，供日后参考。

(1)针对 S3C2410、S3C2440 NAND Flash 控制器的不同来定义一些数据结构和函数，在 include/s3c24x0.h 文件中增加 S3C2440_NAND 数据结构。此处为冗余的操作，为以后升级做准备。

```
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;
} S3C2440_NAND;
```

(2)在 include/s3c2410.h 文件中仿照 S3C2410_GetBase_NAND 函数定义 S3C2440_GetBase_NAND 函数。

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

(3) 实现 cpu/arm920t/s3c24x0/nand_flash.c，即添加 nand_flash.c 文件

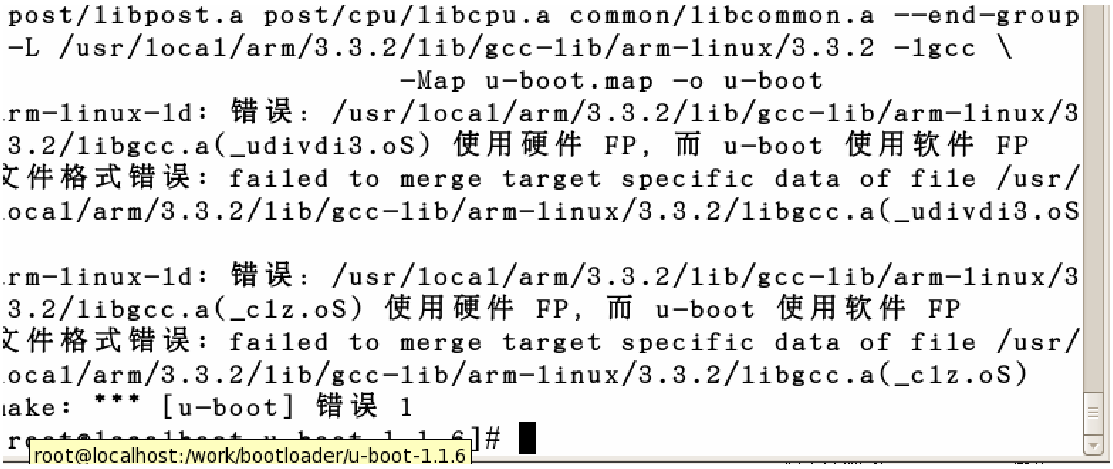
文件内容如下:

```
#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)



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;
    }
}


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;
    }
}


static int s3c2410_nand_devready(struct mtd_info *mtd)
{
    S3C2410_NAND * const s3c2410nand = S3C2410_GetBase_NAND();

    return (s3c2410nand->NFSTAT & S3C2410_NFSTAT_READY);
}



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;
    }
}


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;
    }
}


static int s3c2440_nand_devready(struct mtd_info *mtd)
{
    S3C2440_NAND * const s3c2440nand = S3C2440_GetBase_NAND();

    return (s3c2440nand->NFSTAT & S3C2440_NFSTAT_READY);
}


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)
    {

        s3c2410nand->NFCONF =
(1<<15)|(1<<12)|(1<<11)|(TACLS<<8)|(TWRPH0<<4)|(TWRPH1<<0);
    }
    else
    {

        s3c2440nand->NFCONF = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4);

        s3c2440nand->NFCONT = (1<<4)|(0<<1)|(1<<0);
    }
}


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
```

(4)将 nand_flash.c 编入 u-boot,即修改 cpu/arm920t/s3c24x0/Makefile 文件

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

至此，编译生成 u-boot.bin 并烧入 NAND Flash,启动，便可以引导内核了。



```
post/libpost.a post/cpu/libcpu.a common/libcommon.a --end-group
-L /usr/local/arm/3.3.2/lib/gcc-lib/arm-linux/3.3.2 -lgcc \
                        -Map u-boot.map -o u-boot
rm-linux-ld: 错误: /usr/local/arm/3.3.2/lib/gcc-lib/arm-linux/3
3.2/libgcc.a(_udivdi3.oS) 使用硬件 FP, 而 u-boot 使用软件 FP
文件格式错误: failed to merge target specific data of file /usr/
ocal/arm/3.3.2/lib/gcc-lib/arm-linux/3.3.2/libgcc.a(_udivdi3.oS

rm-linux-ld: 错误: /usr/local/arm/3.3.2/lib/gcc-lib/arm-linux/3
3.2/libgcc.a(_clz.oS) 使用硬件 FP, 而 u-boot 使用软件 FP
文件格式错误: failed to merge target specific data of file /usr/
ocal/arm/3.3.2/lib/gcc-lib/arm-linux/3.3.2/libgcc.a(_clz.oS)
ake: *** [u-boot] 错误 1
r  root@localhost:/work/bootloader/u-boot-1.1.6 ]#
```

注解:

如果移植的过程中，用的交叉编译器为 3.3.2 可能会出现硬浮点与软浮点的问题

解决方式为

修改 cpu/arm920t/config.mk 文件

将:

```
PLATFORM_CPPFLAGS +=$(call cc-option,-mapcs-32,-mabi=apcs-gnu)
```

改为:

```
PLATFORM_CPPFLAGS +=$(call cc-option,-mapcs-32,$(call
cc-option,-mabi=apcs-gnu),)
```

同时将

```
PLATFORM_RELFLAGS += -fno-strict-aliasing  -fno-common -ffixed-r8
#   -msoft-float
```

后面的

```
#   -msoft-float
```

注释掉

Make distclean

Make fs2410_config

Make

注意，此处修改了 config.mk 文件，所以必须 make distclean

四、修改配置文件,针对开发板改变 cpu 主频。

(1)U-BOOT 给 linux 内核传递合适参数的定义

修改 include/configs/fs2410.h 如下:

……

……

```
#define        CONFIG_RTC_S 3C24X0    1

#define CONFIG_ENV_OVERWRITE
#define CONFIG_BAUDRATE        115200
```

添加

```
#define CONFIG_CMDLINE_TAG  1
#define CONFIG_SETUP_MEMORY_TAGS 1
#define CONFIG_INITRD_TAG       1
```

（2）修改 UBOOT 的 2410CPU 频率

smdk2410 的 U-BOOT 原来运行频率是 202.8M，而 FS2410 的 BIOS 里面是 200M，所以不修改频率可能会出点问题。按照网上的说法，内核中，在 \arch\arm\mach_s3c2410\s3c2410.c 中，fclk = s3c2410_get_pll(MPLLCON, xtal); //读出来的 fclk 结果和 bootloader 的频率不一致。

修改 board/fs2410/fs2410.c 文件如下：

```
#define FCLK_SPEED 1
#if FCLK_SPEED==0
#define M_MDIV 0xC3
#define M_PDIV 0x4
#define M_SDIV 0x1
#elif FCLK_SPEED==1
//#define M_MDIV 0xA1
//#define M_PDIV 0x3
//#define M_SDIV 0x1
#define M_MDIV 0x5c
#define M_PDIV 0x4
#define M_SDIV 0x0
#endif
```

五，到现在为止，nand 不能实现对 yaffs 类型文件系统的烧写工作。下面添加 yaffs 文件支持

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...″命令实现 yaffs 文件系统映象的烧写。

(1)在 commom/cmd_nand.c 中增加″nand write.yaffs...″ 的使用说明，代码添加如下：(注意添加的位置)

```
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"
……………………
……………………
```

(2) 然后，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：

```
……………………
……………………
opts.quiet      = quiet;
                ret = nand_write_opts(nand, &opts);
            }
}
else if (  s != NULL && !strcmp(s, ".yaffs")){
            if (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 {

                nand_write_options_t opts;
                memset(&opts, 0, sizeof(opts));
                opts.buffer = (u_char*) addr;
                opts.length = size;
                opts.offset = off;

                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);
        }
…………………
…………………
```

NAND Flash 每一页大小为 (512+16)字节 (还有其他格式的 NAND Flash ，比如每页大小为(256+8)、(2048+64)等，SAMSUNG_K9F1G08U0B 就是(2048+64))，其中 512 字节就是数据存储区，16 字节称为 OOB(Out Of Band)区，在 OBB 区存放坏块标记、前面 512 字节的 ECC 校验码等。

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。

(3)在 include/nand.h 中进行如下修改，增加 skipfirstblk 成员：

```
struct nand_write_options {
u_char *buffer;
………………
………………
int pad;
int blockalign;
int skipfirstblk;
};
```

(4)在 drivers/nand/nand_util.c 修改 nand_write_opts 函数，增加对 skipfirstblk 成员的支持：

```
int nand_write_opts(nand_info_t *meminfo, const nand_write_options_t
*opts)
{
    int imglen = 0;
     ………………
     ………………
    int result;
    int skipfirstblk = opts->skipfirstblk;
    ………………
    ………………

} while (offs < blockstart + erasesize_blockalign);
         }
```

```
        if (skipfirstblk) {
            mtdoffset += erasesize_blockalign;
            skipfirstblk = 0;
            continue;
        }
    readlen = meminfo->oobblock;
………………
```

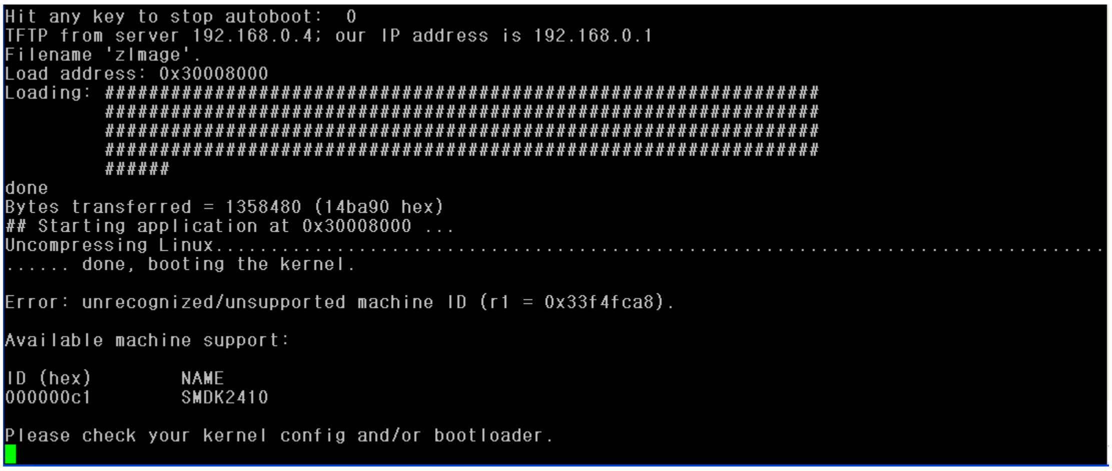
(5)进行上面移植后，u-boot 已经支持 yaffs 文件系统映象的烧写，由于前面设置"opts.noecc=1" 不使用 ECC 校验码，烧写时会提示很多提示信息，可以修改 drivers/nand/nand_base.c 文件中的 nand_write_page 函数，将其注释掉。

```
    case NAND_ECC_NONE:
        //printk (KERN_WARNING "Writing data without ECC to NAND-FLASH is not recommended\n");
```

最后在 u-boot 顶层目录执行：

```
make s3c2410_config
make
```

命令后，在 u-boot 顶层目录中生成 u-boot.bin 文件，用 jtag 线下到板子上 reset 正常启动。



```
Hit any key to stop autoboot:  0
TFTP from server 192.168.0.4; our IP address is 192.168.0.1
Filename 'zImage'.
Load address: 0x30008000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         ######
done
Bytes transferred = 1358480 (14ba90 hex)
## Starting application at 0x30008000 ...
Uncompressing Linux.............................................................................
...... done, booting the kernel.

Error: unrecognized/unsupported machine ID (r1 = 0x33f4fca8).

Available machine support:

ID (hex)        NAME
000000c1        SMDK2410

Please check your kernel config and/or bootloader.
```

六、完善 do_go 函数

编辑 common/cmd_boot.c 函数即可。

```
gedit common/cmd_boot.c
```

添加

```
/*添加 call_linux 函数*/
******************************************************************************
*
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)
);
}
```

****

添加 setup_linux_param 函数

****

```
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);
}
```

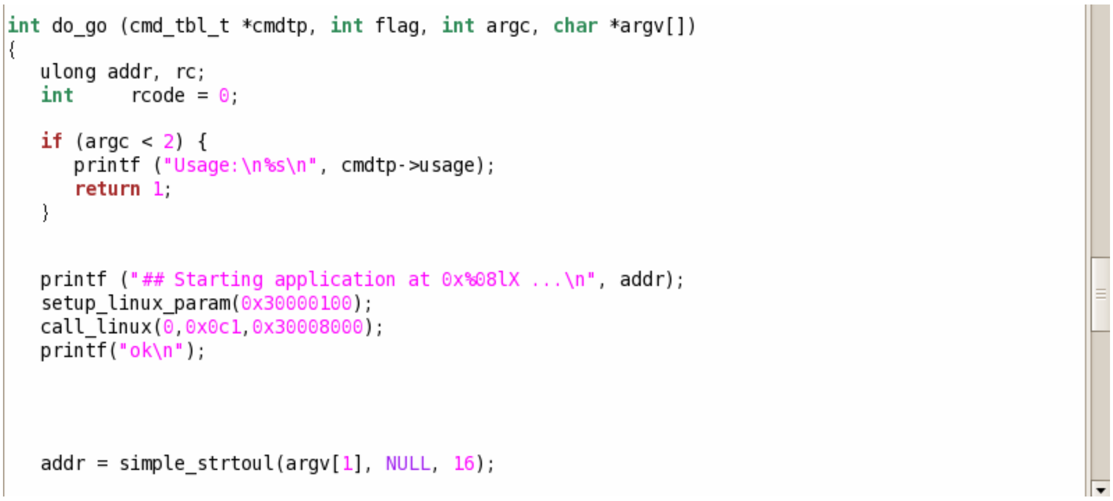
****

在 do_go()函数中添加

****

```
printf ("## Starting application at 0x%08lX ...\n", addr);
         setup_linux_param(0x30000100);
         call_linux(0,0x0c1,0x30008000);
         printf("ok\n");
--------------------------------------------------------------------------------
```

添加位置如图!



```
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;
   }


   printf ("## Starting application at 0x%08lX ...\n", addr);
   setup_linux_param(0x30000100);
   call_linux(0,0x0c1,0x30008000);
   printf("ok\n");



   addr = simple_strtoul(argv[1], NULL, 16);
```

至此 uboot 移植全部完成，能够烧写 yaffs 文件系统是其主要特点。