移植
U-BOOT-2012.07
到 MINI2440

By Charles Yang
September 13th, 2012

今天，开始移植目前最新的 U-BOOT（U-BOOT-2012.07）到 MINI2440 开发板
特此做如下笔记，移植的主要任务如下：

```
###############################################################################
#       1)  移植，创建新的板级目录与文件                                      #
#       2)  移植，设置时钟、内存 SDRAM，和支持串口(UART)显示                  #
#       3)  移植，支持 NANDFLASH 启动                                         #
#       4)  移植，支持 NORFLASH 操作                                          #
#       5)  移植，支持 NANDFLASH 操作（nand read, write, erase 等等）         #
#       6)  移植，支持 DM9000 网卡（ping, tftp, nfs 等）                      #
#       7)  移植，裁剪 U-BOOT、修改设置环境参数和分区表                       #
#       8)  移植，支持 NANDFLASH 烧写 YAFFS 镜像                              #
#       9)  移植，支持启动菜单选项和添加命令                                  #
#      10)  移植，支持内核启动                                                #
#      11)  移植，支持 USB 下载                  (选做)                       #
#      12)  移植，支持 LCD 显示 LOGO             (选做)                       #
#      13)  移植，支持 SD 卡                     (选做)                       #
###############################################################################
```

一、 编译 U-BOOT-2012.07 的前提条件

预安装交叉编译工具链
目前使用的是：arm-linux-gcc-4.3.2.tgz

二、 U-BOOT-2012.07 的下载与初体验

1. 从 ftp://ftp.denx.de/pub/u-boot/ 下载最新的 u-boot-2012.07.tar.bz2

2. 把 u-boot-2012.07.tar.bz2 放到 ubuntu 主机上，执行
   tar xjf u-boot-2012.07.tar.bz2

3. 然后进入 u-boot-2012.07 根目录，执行
   cd u-boot-2012.07

4. 执行
   make smdk2410_config
   make
   make 成功执行后，得到 u-boot.bin

5. 把 u-boot.bin 烧写到 2440 的 NORFLASH 上
   烧写方法两种：
   方法一、用 openJTAG 把编译出来的 u-boot.bin 烧写到 NORFLASH 中。

(不过，由于新的 u-boot 编译出来的 bin 文件很大，所以直接烧写会很慢)，所以，可以考虑使用第二种办法

方法二、用 openJTAG 先烧写老版本移植成功的 u-boot.bin，然后使用串口，usb、或网络等方法下载新的 u-boot.bin

6. 烧好之后，重新上电，发现终端没有一点反应，证明 2410 配置编译出来的 bin 文件不能在 2440 开发板上运行，所以，我们需要修改代码来支持 2440 开发板。

三、 开始移植工作

1) 移植，创建新的板级目录与文件

第一步：由于之前 u-boot 编译过，我们先把生成的.o 以及临时文件都清除掉，在 ubuntu 上的 u-boot-2012.07 根目录下执行：

make distclean

第二步：

a) 创建新的板级目录 smdk2440，在 ubuntu 上的 u-boot-2012.07 根目录下执行：

cp -rf board/samsung/smdk2410/ board/samsung/smdk2440

mv -f board/samsung/smdk2440/smdk2410.c board/samsung/smdk2440/smdk2440.c

vi board/samsung/smdk2440/Makefile

在 Makefile 里：
COBJS := smdk2410.o
修改为
COBJS := smdk2440.o

b) 创建新的板级配置相关的头文件 smdk2440.h，
在 ubuntu 上的 u-boot-2012.07 根目录下执行：

cp -f include/configs/smdk2410.h include/configs/smdk2440.h

c) 修改 u-boot-2012.07 根目录下的 boards.cfg，添加 smdk2440_config，在 ubuntu 上的 u-boot-2012.07 根目录下执行：

vi boards.cfg
复制
smdk2410 arm arm920t – samsung s3c24x0
这一行，在下一行粘贴，并修改为
smdk2440 arm arm920t – samsung s3c24x0
保存退出

d) 在 ubuntu 上的 u-boot-2012.07 根目录下执行：
make smdk2440_config
make
测试是否能编译成功？如果编译成功，证明我们创建的 smdk2440 板级目录和文件已经有效。

2) 移植，设置时钟、内存 SDRAM，和支持串口(UART)显示

- 设置时钟

第一步：在 board/samsung/smdk2440/smdk2440.c 文件中找到 board_early_init_f 函数

```
writel(0xFFFFFF, &clk_power->locktime);
writel((M_MDIV << 12) + (M_PDIV << 4) + M_SDIV,
                    &clk_power->mpllcon);
```

修改为

```
//writel(0xFFFFFF, &clk_power->locktime);
//writel((M_MDIV << 12) + (M_PDIV << 4) + M_SDIV,
//                    &clk_power->mpllcon);
```

第二步：在 arch/arm/cpu/arm920t/start.S 中，添加对时钟的设置

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

修改为

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

- 在#endif 后添加如下代码：

```
#define S3C2440_MPLL_400MHZ
        ((0x5c<<12)|(0x01<<4)|(0x01))

ldr r0, =0x4c000014
mov r1, #0x05;     /* FCLK:HCLK:PCLK=1:4:8 */
str r1, [r0]

/* Setup to asynchronous mode */
mrc p15, 0, r1, c1, c0, 0
orr r1, r1, #0xc0000000
mcr p15, 0, r1, c1, c0, 0

ldr r0, =0x4c000004
ldr r1, =S3C2440_MPLL_400MHZ
str r1, [r0]

/* Enable I-Cache */
mrc p15, 0, r0, c1, c0, 0     @ read control reg
orr r0, r0, #(1<<12)
mcr p15, 0, r0, c1, c0, 0     @ write it back
```

- 设置 SDRAM

第一步：在 board/samsung/smdk2440/lowlevel_init.S 文件中注释掉 SMRDATA：后边的全部代码
替换为以下的代码：

```
.long 0x22011110        //BWSCON
.long 0x00000700        //BANKCON0
.long 0x00000700        //BANKCON1
.long 0x00000700        //BANKCON2
.long 0x00000700        //BANKCON3
.long 0x00000740        //BANKCON4
.long 0x00000700        //BANKCON5
.long 0x00018005        //BANKCON6
.long 0x00018005        //BANKCON7
.long 0x008C04F4        //REFRESH
.long 0x000000B1        //BANKSIZE
.long 0x00000030        //MRSRB6
.long 0x00000030        //MRSRB7
```

第二步：在 ubuntu 上的 u-boot-2012.07 根目录下重新编译 u-boot，执行：

```
make clean
make
```

第三步： 把第二步编译出来的 u-boot.bin 烧到 2440 开发板的 NORFLASH 中，并重新上电。
终端显示有乱码

- 设置串口显示

第一步：在 include/configs/smdk2440.h 文件中
注视掉
//#define CONFIG_S3C2410
//#define CONFIG_CMD_NAND
//#define CONFIG_YAFFS2
添加
#define CONFIG_S3C2440

第二步：重新编译 u-boot，执行
make clean
make

第三步：把编译出来的 u-boot.bin 烧到 2440 开发板的 NORFLASH 中，重新上电，终端有如下显示，说明，开发板已支持 NORFLASH 启动。

```
-----------------------------------------------------------------------
U-Boot 2012.07 (Sep 13 2012 - 23:38:17)

CPUID: 32440001
FCLK:    405.600 MHz
HCLK:    101.400 MHz
PCLK:     50.700 MHz
DRAM:    64 MiB
WARNING: Caches not enabled
Flash: *** failed ***
### ERROR ### Please RESET the board ###
-----------------------------------------------------------------------
```

3) 移植，支持 NANDFLASH 启动

第一步：修改 arch/arm/Config.mk 文件
把倒数第二行的
LDFLAGS_u-boot += -pie
修改为
#LDFLAGS_u-boot += -pie

第二步：把我们写好的 boot_init.c 文件拷贝到 board/samsung/smdk2440 目录下

第三步：修改 board/samsung/smdk2440/Makefile 文件

COBJS := smdk2440.o
修改为
COBJS := smdk2440.o boot_init.o

第四步：修改 arch/arm/cpu/arm920t/start.S 文件

■ 把以下代码注释掉

```
#if 0
/*
 * void relocate_code (addr_sp, gd, addr_moni)
 *
 * This "function" does not return, instead it
   continues in RAM
 * after relocating the monitor code.
 *
*/

.globl   relocate_code
relocate_code:
    mov      r4, r0     /* save addr_sp */
    mov      r5, r1     /* save addr of gd */
    mov      r6, r2     /* save addr of destination */

    /* Set up the stack */
stack_setup:
    mov      sp, r4
    adr  r0, _start
    cmp      r0, r6
    beq clear_bss      /* skip relocation */
    mov      r1, r6     /* r1 <- scratch for copy_loop */
    ldr   r3, _bss_start_ofs
    add r2, r0, r3      /* r2 <- source end address */

copy_loop:
    ldmia    r0!, {r9-r10}
    stmia    r1!, {r9-r10}
    cmp      r0, r2
    blo  copy_loop

#ifndef CONFIG_SPL_BUILD
```

```
    /*
     * fix .rel.dyn relocations
     */
    ldr   r0, _TEXT_BASE        /* r0 <- Text base */
    sub r9, r6, r0              /* r9 <- relocation offset */
    ldr   r10, _dynsym_start_ofs
    add r10, r10, r0   /* r10 <- sym table in FLASH */
    ldr   r2, _rel_dyn_start_ofs
    add r2, r2, r0
    ldr   r3, _rel_dyn_end_ofs
    add r3, r3, r0      /* r3 <- rel dyn end in FLASH */

fixloop:
    ldr   r0, [r2]
    add r0, r0, r9
    ldr   r1, [r2, #4]
    and r7, r1, #0xff
    cmp r7, #23               /* relative fixup? */
    beq fixrel
    cmp         r7, #2                /* absolute fixup? */
    beq fixabs
    /* ignore unknown type of fixup */
    b     fixnext

fixabs:
    mov r1, r1, LSR #4
    add r1, r10, r1
    ldr   r1, [r1, #4]          /* r1 <- symbol value */
    add r1, r1, r9       /* r1 <- relocated sym addr */
    b fixnext

fixrel:
    /* relative fix: increase location by offset */
    ldr   r1, [r0]
    add r1, r1, r9

fixnext:
    str   r1, [r0]
    add r2, r2, #8
    cmp       r2, r3
    blo  fixloop
#endif

clear_bss:
```

```
#ifndef CONFIG_SPL_BUILD
    ldr   r0, _bss_start_ofs
    ldr   r1, _bss_end_ofs
    mov       r4, r6                /* reloc addr */
    add r0, r0, r4
    add r1, r1, r4
    mov r2, #0x00000000       /* clear */

clbss_l:cmp       r0, r1                /* clear loop... */
    bhs clbss_e         /* if reached end of bss, exit */
    str   r2, [r0]
    add r0, r0, #4
    b     clbss_l

clbss_e:
    bl coloured_LED_init
    bl red_led_on
#endif
#endif
```

■ 在以下代码后添加程序

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

添加如下代码

```
/* Setup the temporary stack */
    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
    bic sp, sp, #7 /* 8-byte alignment for ABI
                          compliance */

    bl    nand_init_ll

    mov r0, #0
    ldr r1, _TEXT_BASE
    /* _bss_start_ofs = the size of u-boot */
    ldr r2, _bss_start_ofs

    bl    copy2ram

    bl    clear_bss

    /* Jump from SRAM to SDRAM */
    ldr pc, =call_board_init_f
```

- 注释掉 call_board_init_f:后的两行

```
#if 0
    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
    bic sp, sp, #7 /* 8-byte alignment for ABI
                      compliance */
#endif
```

- 在

```
.globl IRQ_STACK_START_IN
IRQ_STACK_START_IN:
    .word    0x0badc0de
```

后边添加如下代码

```
.globl stack_base
stack_base:
    .long 0
```

- 在 bl board_init_f 后添加

```
/* The return value of board_init_f is stored in r0 */
ldr r1, _TEXT_BASE
/* Reallocate the stack pointer */
ldr sp, stack_base

/* Invoking SECOND STAGE */
bl board_init_r
```

- 注释掉如下代码

```
#if 0
    #ifdef CONFIG_NAND_SPL
    ldr         r0, _nand_boot_ofs
    mov       pc, r0

    _nand_boot_ofs:
          .word nand_boot
    #else
    ldr   r0, _board_init_r_ofs
    adr  r1, _start
    add lr, r0, r1
    add lr, lr, r9
    /* setup parameters for board_init_r */
    mov       r0, r5          /* gd_t */
    mov       r1, r6          /* dest_addr */
    /* jump to it ... */
    mov       pc, lr
```

```
_board_init_r_ofs:
    .word board_init_r - _start
#endif

_rel_dyn_start_ofs:
    .word __rel_dyn_start - _start
_rel_dyn_end_ofs:
    .word __rel_dyn_end - _start
_dynsym_start_ofs:
    .word __dynsym_start - _start
#endif
```

到此为止，start.S 文件就修改完成。

第五步：修改 arch/arm/lib/board.c 文件

- 修改函数 board_init_f 的返回类型

  void board_init_f(ulong bootflag)

  修改为

  unsigned long board_init_f(ulong bootflag)

- 在 ulong addr, addr_sp;之后添加

  extern ulong stack_base;

- 注释掉

```
#if 0
    addr -= gd->mon_len;
    addr &= ~(4096 - 1);
#endif
```

  在后面添加

```
addr = CONFIG_SYS_TEXT_BASE;
```

- 注释掉

```
#if 0
    relocate_code(addr_sp, id, addr);
#endif
```

- 在 relocate_code 之前，添加

  stack_base = addr_sp;

- 在 relocate_code 之后，添加

  return (unsigned long)id;

第六步：修改 include/common.h 文件

void board_init_f (ulong) __attribute__ ((noreturn));
修改为
unsigned long board_init_f (ulong) ;

第七步：修改 arch/arm/cpu/U-boot.lds 文件

在 CPUDIR/start.o (.text) 后添加
board/samsung/smdk2440/libsmdk2440.o (.text)

第八步：修改 include/configs/smdk2440.h 文件

#define CONFIG_SYS_TEXT_BASE 0x0
修改为
#define CONFIG_SYS_TEXT_BASE 0x33F00000

第九步：重新编译 u-boot

make clean
make

第十步：把 2440 开发板的启动方式设置为 NANDFLASH 启动，然后烧写新的 u-boot.bin 到 2440 开发板的 NANDFLASH 中，烧写完之后，重启开发板，可以看到终端有如下显示，证明 NANDFLASH 启动已经成功。

U-Boot 2012.07 (Sep 14 2012 - 11:40:58)

CPUID: 32440001
FCLK: 405.600 MHz
HCLK: 101.400 MHz
PCLK: 50.700 MHz
DRAM: 64 MiB
WARNING: Caches not enabled
Flash: * failed ****
ERROR ### Please RESET the board
---

4) 移植，支持 NORFLASH 操作

第一步：修改 arch/arm/lib/board.c 文件

puts(failed);
hang();
修改为
//puts(failed);
//hang();

在 puts(failed)之前添加
puts("0 KB\r\n");

第二步：修改 drivers/mtd/Jedec_flash.c 文件

在 jedec_table 数组的最后一行添加如下代码：

```
{
    .mfr_id = (u16)SST_MANUFACT,    /* should be CFI */
   .dev_id    = SST39VF1601,
    .name    = "SST 39VF1601",
     .uaddr    = {
        [0] = MTD_UADDR_0x5555_0x2AAA,    /* x8 */
        [1] = MTD_UADDR_0x5555_0x2AAA      /* x16 */
     },
     .DevSize    =    SIZE_2MiB,
     .CmdSet    =    P_ID_AMD_STD,
     .NumEraseRegions = 2,
     .regions    =    {
              ERASEINFO(0x1000,256),
              ERASEINFO(0x1000,256)
     }
},
```

第三步：修改 include/configs/smdk2440.h 文件

#define CONFIG_SYS_MAX_FLASH_SECT   (19)
修改为
#define CONFIG_SYS_MAX_FLASH_SECT   (512)

第四步：把 2440 开发板的启动方式调到 NORFLASH 启动，重新编译 u-boot，把编译好的 u-boot.bin 烧写到 2440 开发板的 NORFLASH 中。终端显示如下：

```
---------------------------------------------------------------------
U-Boot 2012.07 (Sep 14 2012 - 13:50:42)

CPUID: 32440001
```

FCLK: 405.600 MHz
HCLK: 101.400 MHz
PCLK: 50.700 MHz
DRAM: 64 MiB
WARNING: Caches not enabled
Flash: 2 MiB
***** Warning - bad CRC, using default environment**

In: serial
Out: serial
Err: serial
Net: CS8900-0
Pete&Charles #

从显示信息可以看出，2440 开发板已经识别出 NORFLASH 为 2 MiB。

5) 移植，支持 NANDFLASH 操作

第一步：修改 include/configs/smdk2440.h 文件

//#define CONFIG_CMD_NAND
修改为
#define CONFIG_CMD_NAND

第二步：修改 drivers/mtd/nand/S3c2410_nand.c 文件

- 在 s3c2410_hwcontrol 函数中

  struct s3c2410_nand *nand =
  s3c2410_get_base_nand();
  修改为
  struct s3c2440_nand *nand =
  s3c2440_get_base_nand();

- 在 s3c2410_dev_ready 函数中

  struct s3c2410_nand *nand =
  s3c2410_get_base_nand();
  修改为
  struct s3c2440_nand *nand =
  s3c2440_get_base_nand();

- 在 board_nand_init 函数中

a) struct s3c2410_nand *nand_reg = s3c2410_get_base_nand();
修改为
struct s3c2440_nand *nand_reg = s3c2440_get_base_nand();

b) cfg = S3C2410_NFCONF_EN;
cfg |= S3C2410_NFCONF_TACLS(tacls - 1);
cfg |= S3C2410_NFCONF_TWRPH0(twrph0 - 1);
cfg |= S3C2410_NFCONF_TWRPH1(twrph1 - 1);
修改为
#ifdef CONFIG_S3C2410
cfg = S3C2410_NFCONF_EN;
cfg |= S3C2410_NFCONF_TACLS(tacls - 1);
cfg |= S3C2410_NFCONF_TWRPH0(twrph0 - 1);
cfg |= S3C2410_NFCONF_TWRPH1(twrph1 - 1);

#ifdef CONFIG_S3C2440
cfg = S3C2440_NFCONF_TACLS(tacls - 1);
cfg |= S3C2440_NFCONF_TWRPH0(twrph0 - 1);
cfg |= S3C2440_NFCONF_TWRPH1(twrph1 - 1);
#endif

并在本文件的开头添加下面 3 个宏
#define S3C2440_NFCONF_TACLS(x)
((x)<<12)
#define S3C2440_NFCONF_TWRPH0(x)
((x)<<8)
#define S3C2440_NFCONF_TWRPH1(x)
((x)<<4)

c) 添加 NANDFLASH NFCONT 寄存器的配置
在 writel(cfg, &nand_reg->nfconf);
后面添加如下代码
cfg = (1<<4)|(1<<1)|(1<<0);
writel(cfg, &nand_reg->nfcont);

d) 添加 NANDFLASH 片选函数的操作
nand->select_chip = NULL;
修改为
nand->select_chip = s3c2440_nand_select_chip;

在 board_nand_init 函数前，添加对
s3c2440_nand_select_chip 函数的实现，代码如

下

```
static void s3c2440_nand_select_chip(struct
        mtd_info *mtd, int chipnr)
{
        struct s3c2440_nand *nand =
                s3c2440_get_base_nand();
    switch (chipnr) {
    case -1:
        writel(readl(&nand->nfcont)  |  (1<<1),
                        &nand->nfcont);
        break;
    case 0:
        writel(readl(&nand->nfcont)  &  ~(1<<1),
                    &nand->nfcont);
        break;

    default:
        BUG();
}
```

■ 修改 s3c2410_hwcontrol 函数为如下代码：

```
static void s3c2410_hwcontrol(struct mtd_info *mtd,
int data, unsigned int ctrl)
{
   struct        s3c2440_nand        *nand        =
s3c2440_get_base_nand();

     if (ctrl & NAND_CLE) {
          /* Send out command */
          writeb(data, &nand->nfcmd);
     }else if (ctrl & NAND_ALE) {
          /* Send out address */
          writeb(data, &nand->nfaddr);
     }
}
```

第三步：重新编译 u-boot，把生成的 u-boot.bin 烧到开发板的 NORFLASH 或者 NANDFLASH 中。

如果是 NORFLASH 启动，终端显示如下：

```
-----------------------------------------------------------------------
U-Boot 2012.07 (Sep 14 2012 - 20:54:29)
```

CPUID: 32440001
FCLK: 405.600 MHz
HCLK: 101.400 MHz
PCLK: 50.700 MHz
DRAM: 64 MiB
WARNING: Caches not enabled
Flash: 2 MiB
NAND: 256 MiB
***** Warning - bad CRC, using default environment**

In: serial
Out: serial
Err: serial
Net: CS8900-0
Pete&Charles #

如果是 NANDFLASH 启动，终端显示如下：

U-Boot 2012.07 (Sep 14 2012 - 20:54:29)

CPUID: 32440001
FCLK: 405.600 MHz
HCLK: 101.400 MHz
PCLK: 50.700 MHz
DRAM: 64 MiB
WARNING: Caches not enabled
Flash: 0 KB
NAND: 256 MiB
***** Warning - bad CRC, using default environment**

In: serial
Out: serial
Err: serial
Net: CS8900-0
Pete&Charles #

从 NANDFLASH 启动的时候，2440 会把 NANDFLASH 中前 4K 的代码自动搬移到片内的 4K 内存(SRAM)中，这种启动方式，NORFLASH 不会被识别出来，所以显示 0 KB。

从显示信息里我们可以看到 NANDFLASH 已经被识别出来。
测试 NANDFLASH 命令：

从 NORFLASH 启动，执行

nand erase 0 80000

nand write 0 0 80000　(第一个 0 是 NORFLASH 的 0 地址，第二个 0 是 NANDFLASH 的 0 地址)

测试 nand write 是否成功，执行如下操作

nand read 30000000 0 80000

cmp.b 30000000 0 80000

如果打印类似如下的信息，说明 NANDFLASH 移植成功

Total of 524288 byte(s) were the same

6) 移植，支持 DM9000 网卡（ping, tftp, nfs 等）

第一步：修改 board/samsung/smdk2440/Smdk2440.c

在 board_eth_init 函数中，添加如下代码

```
#ifdef    CONFIG_DRIVER_DM9000
    rc = dm9000_initialize(bis);
#endif
```

第二步：修改 include/configs/smdk2440.h

```
#define CONFIG_CS8900
#define CONFIG_CS8900_BASE   0x19000300
#define CONFIG_CS8900_BUS16
```

修改为

```
#if 0
    #define CONFIG_CS8900
    #define CONFIG_CS8900_BASE     0x19000300
    #define CONFIG_CS8900_BUS16
#endif
```

添加如下代码：

```
#define CONFIG_DRIVER_DM9000
#define CONFIG_DM9000_BASE              (0x20000000)
#define DM9000_IO            (CONFIG_DM9000_BASE)
#define DM9000_DATA       (CONFIG_DM9000_BASE + 4)
```

第三步：修改 drivers/net/Dm9000x.c 文件

在 dm9000_init 函数中，注释掉以下代码：

```
i = 0;
while (!(dm9000_phy_read(1) & 0x20)) {
    udelay(1000);
    i++;
    if (i == 10000) {
```

```
            printf("could not establish link\n");
            return 0;
        }
    }
```

修改为

```
#if 0
    i = 0;
    while (!(dm9000_phy_read(1) & 0x20)) {
        udelay(1000);
        i++;
        if (i == 10000) {
            printf("could not establish link\n");
            return 0;
        }
    }
#endif
```

第四步：重新编译 u-boot，把新编译出来的 u-boot.bin 烧到 NORFLASH 中，重启开发板，终端显示如下：

```
---------------------------------------------------------------------
U-Boot 2012.07 (Sep 14 2012 - 20:54:29)

CPUID: 32440001
FCLK:    405.600 MHz
HCLK:    101.400 MHz
PCLK:     50.700 MHz
DRAM:    64 MiB
WARNING: Caches not enabled
Flash:     0 KB
NAND:    256 MiB
*** Warning - bad CRC, using default environment

In:       serial
Out:      serial
Err:      serial
Net:      dm9000
Pete&Charles #
---------------------------------------------------------------------
```

使用 ping，tftp 和 nfs 命令测试 DM9000 网卡。

由于我的虚拟机 ubuntu 的 ip 地址是 192.168.1.102，在开发板上执行 **print** 之后，显示如下：

baudrate=115200
bootdelay=5
ethact=dm9000
ipaddr=10.0.0.110
netmask=255.255.255.0
serverip=10.0.0.1
stderr=serial
stdin=serial
stdout=serial

Environment size: 160/65532 bytes

在开发板上执行：

i) set ipaddr 192.168.1.230 (开发板 ip 地址)
ii) set ethaddr 1a:2b:3c:4d:5e:6f (开发板 MAC 地址)
iii) ping 192.168.1.102 (虚拟机 ubuntu ip 地址)

显示如下信息说明已经 ping 通虚拟机 ubuntu。

dm9000 i/o: 0x20000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 1a:2b:3c:4d:5e:6f
operating at 100M full duplex mode
Using dm9000 device
host 192.168.1.102 is alive

7) 移植，裁剪 U-BOOT、修改设置环境参数和分区表

- 裁剪 U-BOOT，现在的 U-BOOT 大概在 370KB 到 400KB 之间
  第一步：修改 include/configs/smdk2440.h 文件
  注释掉以下代码：

```
#define CONFIG_USB_OHCI
#define CONFIG_USB_KEYBOARD
#define CONFIG_USB_STORAGE
#define CONFIG_DOS_PARTITION
#define CONFIG_RTC_S3C24X0
#define CONFIG_CMD_BSP
#define CONFIG_CMD_CACHE
#define CONFIG_CMD_DATE
#define CONFIG_CMD_DHCP
#define CONFIG_CMD_ELF
#define CONFIG_CMD_REGINFO
#define CONFIG_CMD_USB
#define CONFIG_CMD_FAT
```

#define CONFIG_CMD_EXT2
#define CONFIG_CMD_UBI
#define CONFIG_CMD_UBIFS
#define CONFIG_YAFFS2
#define CONFIG_RBTREE
修改为
//#define CONFIG_USB_OHCI
//#define CONFIG_USB_KEYBOARD
//#define CONFIG_USB_STORAGE
//#define CONFIG_DOS_PARTITION
//#define CONFIG_RTC_S3C24X0
//#define CONFIG_CMD_BSP
//#define CONFIG_CMD_CACHE
//#define CONFIG_CMD_DATE
//#define CONFIG_CMD_DHCP
//#define CONFIG_CMD_ELF
//#define CONFIG_CMD_REGINFO
//#define CONFIG_CMD_USB
//#define CONFIG_CMD_FAT
//#define CONFIG_CMD_EXT2
//#define CONFIG_CMD_UBI
//#define CONFIG_CMD_UBIFS
//#define CONFIG_YAFFS2
//#define CONFIG_RBTREE

第二步：重新编译 U-BOOT，查看编译好的 u-boot.bin，在 ubuntu 虚拟机的 u-boot 根目录下执行：

ls –l u-boot.bin

经过裁剪后的 u-boot.bin 大概在 215KB 左右。

- 修改设置环境参数和分区表

第一步：修改 include/configs/Smdk2440.h 文件

- 添加如下代码：

#define CONFIG_ETHADDR ff:ff:ff:ff:ff:ff
#define CONFIG_ENV_IS_IN_NAND
#define CONFIG_ENV_OFFSET 0x40000
#define CONFIG_ENV_SIZE 0x20000
#define CONFIG_ENV_RANGE 0x20000

- 修改如下代码

#define CONFIG_IPADDR 10.0.0.110

```
#define CONFIG_SERVERIP        10.0.0.1
```
修改为
```
#define CONFIG_IPADDR          192.168.1.230
#define CONFIG_SERVERIP        192.168.1.101

#define CONFIG_ENV_ADDR
            (CONFIG_SYS_FLASH_BASE + 0x070000)
#define CONFIG_ENV_IS_IN_FLASH
#define CONFIG_ENV_SIZE            0x10000
```
修改为
```
//#define CONFIG_ENV_ADDR
            (CONFIG_SYS_FLASH_BASE + 0x070000)
//#define CONFIG_ENV_IS_IN_FLASH
//#define CONFIG_ENV_SIZE              0x10000
```

第二步：修改分区表

修改 include/configs/Smdk2440.h 文件

添加如下代码：

```
#define MTDIDS_DEFAULT
            "nand0=Pete&Charles-0"

#define MTDPARTS_DEFAULT
        "mtdparts=Pete&Charles-0:256k(u-boot)," \
         "128k(params)," \
         "2m(kernel)," \
         "-(root)"
```

第三步：修改 arch/arm/lib/Board.c 文件

在

```
/* main_loop() can return to retry autoboot, if so
        just run it again. */
for (;;) {
    main_loop();
}
```

前面添加如下代码：

```
run_command("mtdparts default", 0);
```

8) 移植，支持 NANDFLASH 烧写 YAFFS 镜像

第一步：修改 include/configs/Smdk2440.h 文件

添加如下代码：

```
#define CONFIG_CMD_NAND_YAFFS
```

第二步：修改 common/Cmd_nand.c 文件
在 do_nand 函数中，大约 669 行左右

```
ret = nand_write_skip_bad(nand, off, &rwsize,
                (u_char *)addr,
                WITH_INLINE_OOB);
```

修改为

```
ret = nand_write_skip_bad(nand, off, &rwsize,
                (u_char *)addr,
                WITH_YAFFS_OOB);
```

第三步：修改 drivers/mtd/nand/Nand_util.c 文件
在 nand_write_skip_bad 函数中，大约 556 行左右

```
ops.mode = MTD_OOB_AUTO;
```

修改为

```
ops.mode = MTD_OOB_RAW;
```

9) 移植，支持启动菜单选项和添加命令

代码参考 **cmd_menu.c** 文件

第一步：把 cmd_menu.c 文件拷贝到 common 目录下。

第二步：修改 common/Makefile 文件
在 **COBJS := $(sort $(COBJS-y))**之前的任意地方添加如下代码：

```
COBJS-y += cmd_menu.o
```

第三步：修改 common/Main.c 文件
在

```
/*
 * Main Loop for Monitor Command Processing
 */
```

之前添加如下代码：(大约 401 行左右)

```
run_command("menu", 0);
```

第四步：重新编译 U-BOOT，把新的 u-boot.bin 烧到开发板上，重新启动，就会有菜单提示。

10) 移植，支持内核启动

第一步：修改 include/configs/Smdk2440.h 文件
添加如下代码：

#define CONFIG_BOOTARGS
"noinitrd console=ttySAC0,115200 root=/dev/mtdblock3"

#define CONFIG_BOOTCOMMAND "nand read
30000000 kernel;bootm 30000000"

如果没有 CONFIG_CMDLINE_TAG、
CONFIG_SETUP_MEMORY_TAGS 和 CONFIG_INITRD_TAG
宏，则需要添加以下代码：

#define CONFIG_CMDLINE_TAG
#define CONFIG_SETUP_MEMORY_TAGS
#define CONFIG_INITRD_TAG

第二步：修改 board/samsung/smdk2440/Smdk2440.c 文件
在 board_init 函数中，大约 116 行左右
根据内核配置的启动编号，在 U-BOOT 这里设置成与内核一致的编号。

gd->bd->bi_arch_number = MACH_TYPE_SMDK2410;
修改为
gd->bd->bi_arch_number = 与内核一致的编号;

MACH_TYPE_开头的机器码可以在 U-BOOT 的
arch/arm/include/asm/Mach-types.h 文件中找到或
定义。

11) 移植，支持 USB 下载 (选作)

12) 移植，支持 LCD 显示 LOGO (选作)

13) 移植，支持 SD 卡 (选作)

四、 总结