到 http://www.linuxidc.com 下载 u-boot-2014.04.tar.bz2

解压后在 board/samsung/目录下仍然没有 2440,虽然没有直接支持 2440 开发板,但其代码已经支持,只需添加相关配置即可。

一、 首先建立自己的开发板

拷贝 board/samsung/smdk2410/目录到 board/tq2440

pengdl@debian:~/work/tq2440/u-boot-2014.04$ cp -a board/samsung/smdk2410/ board/tq2440

进入 board/tq2440 目录修改

pengdl@debian:~/work/tq2440/u-boot-2014.04/board/tq2440$ mv smdk2410.c tq2440.c

修改该目录下的 Makefile

COBJS := tq2440.o

拷贝配置文件(使用相似的 smdk2410 开发板的配置文件)

pengdl@debian:~/work/tq2440/u-boot-2014.04$ cp include/configs/smdk2410.h include/configs/tq2440.h

增加开发板配置选项

在顶层目录下的 Makefile 中搜索不到 smdk2410

在顶层目录执行如下命令

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "smdk2410" * -nR

./boards.cfg:74:

smdk2410 arm arm920t - samsung

s3c24x0

./MAINTAINERS:774:smdk2410 ARM920T

./board/samsung/smdk2410/Makefile:28:COBJS := smdk2410.o

./board/tq2440/Makefile:28:COBJS := smdk2410.o

./arch/arm/include/asm/mach-types.h:1646:# define machine_is_smdk2410() (machine_arch_type == MACH_TYPE_SMDK2410)

./arch/arm/include/asm/mach-types.h:1648:# define machine_is_smdk2410() (0)

从这里知道在顶层目录下的 boards.cfg 文件中定义了 smdk2410 开发板的配置选项，仿照它定义 TQ2440 开发板的配置选项

Status, Arch, CPU:SPLCPU, SoC, Vendor, Board name, Target, Options, Maintainers

##

Active arm arm920t s3c24x0 - tq2440

tq2440 -

-

由于我在 board 目录下创建开发板目录，所以 Vendor 指定为空

二、 配置时钟

先大致看一下配置文件 include/configs/tq2440.h

```
/*
 * High Level Configuration Options
 * (easy to change)
 */
#define CONFIG_ARM920T      /* This is an ARM920T Core */
#define CONFIG_S3C24X0      /* in a SAMSUNG S3C24x0-type SoC */
#define CONFIG_S3C2410      /* specifically a SAMSUNG S3C2410 SoC */
#define CONFIG_SMDK2410     /* on a SAMSUNG SMDK2410 Board */
```

这里是高级别的一些配置，配置了 S3C2410 SoC 和 SMDK2410 Board，跟我使用的开发板不一致

根据我自己的开发板 tq2440 进行如下配置

```
//#define CONFIG_S3C2410            /* specifically a SAMSUNG S3C2410 SoC */
#define CONFIG_S3C2440
//#define CONFIG_SMDK2410           /* on a SAMSUNG SMDK2410 Board */
#define CONFIG_AUTO_COMPLETE   //开启命令自动补全
#define CONFIG_SYS_PROMPT "TQ2440 # "       //  命令提示符
屏蔽一些暂时不用的支持，用的时候再加上
#if 0
#define CONFIG_CS8900      /* we have a CS8900 on-board */
#define CONFIG_CS8900_BASE 0x19000300
#define CONFIG_CS8900_BUS16     /* the Linux driver does accesses as shorts */
#endif
#if 0
#define CONFIG_USB_OHCI
#define CONFIG_USB_KEYBOARD
#define CONFIG_USB_STORAGE
#define CONFIG_DOS_PARTITION
#endif
//#define CONFIG_CMD_DHCP
//#define CONFIG_CMD_NAND
//#define CONFIG_CMD_PING
//#define CONFIG_CMD_REGINFO
//#define CONFIG_CMD_USB
#if 0
#define CONFIG_CMD_FAT
#define CONFIG_CMD_EXT2
```

```
#define CONFIG_CMD_UBI
#define CONFIG_CMD_UBIFS
#define CONFIG_CMD_MTDPARTS
#define CONFIG_MTD_DEVICE
#define CONFIG_MTD_PARTITIONS
#define CONFIG_YAFFS2
#define CONFIG_RBTREE
#endif
```

看下链接脚本 arch/arm/cpu/u-boot.lds

```
CPUDIR/start.o (.text*)
```

从这里可以知道 u-boot 执行的第一个文件是 arch/arm/cpu/arm920t/start.S

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

上面几行代码是针对 S3C2410 的

添加时钟初始化代码如下

```
# if defined(CONFIG_S3C2410)
 ldr  r1, =0x3ff
 ldr  r0, =INTSUBMSK
 str  r1, [r0]
# endif
# if defined(CONFIG_S3C2440)
 ldr  r1, =0x7fff
```

```
 ldr  r0, =INTSUBMSK     //  屏蔽子中断
 str  r1, [r0]
# endif /* CONFIG_S3C2440 */
# if defined(CONFIG_S3C2440)
# define MPLLCON    0x4C000004   //系统主频配置寄存器
# define UPLLCON    0x4C000008   //USB 频率配置寄存器
# define CAMDIVN    0x4C000018   //照相机时钟分频寄存器
 ldr r0, =CAMDIVN
 mov r1, #0
 str r1, [r0]
 ldr r0, =CLKDIVN
 mov r1, #0x05
 str r1, [r0]
 /*如果 HDIVN 不等于 0，CPU 必须设置为异步总线模式*/
 mrc p15,0,r0,c1,c0,0
 orr r0,r0,#0xc0000000
 mcr p15,0,r0,c1,c0,0
 ldr r0, =UPLLCON
 ldr r1, =0x38022    // USB 时钟 48MHZ
 str r1, [r0]
 /*
 **When you set MPLL&UPLL values, you have to set the UPLL
 **value first and then the MPLL value. (Needs intervals
 **approximately 7 NOP)
 */
```

```
 nop
 nop
 nop
 nop
 nop
 nop
 nop
 ldr r0, =MPLLCON
 ldr r1, =0x5c011 //CPU 时钟 400MHZ
 str r1, [r0]
# else
 /* FCLK:HCLK:PCLK = 1:2:4 */
 /* default FCLK is 120 MHz ! */
 ldr  r0, =CLKDIVN
 mov     r1, #3
 str  r1, [r0]
#endif /* CONFIG_S3C2440 */
#endif   /* CONFIG_S3C24X0 */
```

board/tq2440/tq2440.c 中 board_early_init_f()函数也初始化了时钟，因为我在 start.S 中已初始化了时钟，所以屏蔽掉 board_early_init_f()中对时钟的初始化代码

```
//     struct s3c24x0_clock_power * const clk_power =
//                             s3c24x0_get_base_clock_power();
 struct s3c24x0_gpio * const gpio = s3c24x0_get_base_gpio();
#if 0
 /* to reduce PLL lock time, adjust the LOCKTIME register */
```

```
 writel(0xFFFFFF, &clk_power->locktime);

 /* configure MPLL */
 writel((M_MDIV << 12) + (M_PDIV << 4) + M_SDIV,
        &clk_power->mpllcon);

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

 /* configure UPLL */
 writel((U_M_MDIV << 12) + (U_M_PDIV << 4) + U_M_SDIV,
        &clk_power->upllcon);

 /* some delay between MPLL and UPLL */
 pll_delay(8000);
#endif
```

可以先配置 u-boot 支持直接烧写进内存 SDRAM 运行

修改配置文件 tq2440.h

#define CONFIG_SYS_TEXT_BASE 0x32000000

CONFIG_SYS_TEXT_BASE 指定了代码的加载地址，待会编译好后生成可执行二进制文件 u—boot.bin，就要把 u-boot.bin 下载到该地址

我们现在需要直接烧写进内存运行，而底层初始化代码还没移植，所以我们需要跳过底层初始化

查看 arch/arm/cpu/arm920t/start.S

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
 bl   cpu_init_crit
```

#endif

如果没有定义 CONFIG_SKIP_LOWLEVEL_INIT 就跳转 cpu_init_crit 函数执行，该函数进行了一些底层的初始化，比如内存。因为下面我们直接将 u-boot 下载到内存中运行，如果在内存中运行的同时再初始化内存，那么内存中的数据代码会遭到破坏。

所以我们在配置文件 tq2440.h 中定义该宏

#define CONFIG_SKIP_LOWLEVEL_INIT

修改一下顶层 Makefile：

202 CROSS_COMPILE ?= arm-linux-

ARCH 就不用我们手动配置了，因为在执行 make tq2440_config 时，会解析 boards.cfg 文件，得到 ARCH 的值。

pengdl@debian:~/work/tq2440/u-boot-2014.04$ make tq2440_config

编译完成生成可执行二进制文件 u—boot.bin,开发板启动原有好的 u-boot

将 u-boot.bin 下载到 SDRAM 的 0x32000000 地址，然后 go 0x32000000 运行

EmbedSky> tftp 0x32000000 u-boot.bin

(注意：在新版本的 u-boot 中，进行代码重定位 relocate 之前就已经在 board_init_f 中调用了全局性的代码，这些全局性的代码并不是位置无关的，所以这里需要将 u-boot 直接下载到它的链接地址处，防止程序跑飞)

dm9000 i/o: 0x20000300, id: 0x90000a46

MAC: 0a:1b:2c:3d:4e:5f

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x32000000

Loading: checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

T ############

done

Bytes transferred = 153844 (258f4 hex)

EmbedSky> go 0x32000000

（注意：必须与刚才 tftp 下载到的地址相同）

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 28 2014 - 23:11:32)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

在 tq2440.h 中定义 DEBUG 宏

#define DEBUG

EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000

dm9000 i/o: 0x20000300, id: 0x90000a46

MAC: 0a:1b:2c:3d:4e:5f

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x32000000

Loading: checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

T ############

done

Bytes transferred = 168860 (2939c hex)

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 29 2014 - 02:51:27)

U-Boot code: 32000000 -> 320246BC BSS: -> 3202A0A4

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

monitor len: 0002A0A4

ramsize: 04000000

TLB table from 33ff0000 to 33ff4000

Top of RAM usable for U-Boot at: 33ff0000

Reserving 168k for U-Boot at: 33fc5000

Reserving 4160k for malloc() at: 33bb5000

Reserving 32 Bytes for Board Info at: 33bb4fe0

Reserving 160 Bytes for Global Data at: 33bb4f40

New Stack Pointer is: 33bb4f30

RAM Configuration:

Bank #0: 30000000 64 MiB

relocation Offset is: 01fc5000

WARNING: Caches not enabled

monitor flash len: 0002939C

Now running in RAM - U-Boot at: 33fc5000

经过加打印，问题定位在 board.c 中的 board_init_r 在调用 mem_malloc_init 函数时出了问题，他完成的操作是将 malloc_start 标识的 malloc 区域清零，这里 malloc 区域的大小是 4MB+160KB，发现在清除到 2MB 多的时候程序就挂了。

这个问题的原因好没有找到，等待解决。目前临时的解决办法是将 malloc 区域的大小减小为 2MB+160KB，做法是修改 tq2440.h 中，将

#define CONFIG_SYS_MALLOC_LEN (4 * 1024 * 1024)

改为：

#define CONFIG_SYS_MALLOC_LEN (2 * 1024 * 1024)

然后编译运行，打印信息如下：

U-Boot code: 32000000 -> 320246BC BSS: -> 3202A0A4

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

monitor len: 0002A0A4

ramsize: 04000000

TLB table from 33ff0000 to 33ff4000

Top of RAM usable for U-Boot at: 33ff0000

Reserving 168k for U-Boot at: 33fc5000

Reserving 2112k for malloc() at: 33db5000

Reserving 32 Bytes for Board Info at: 33db4fe0

Reserving 160 Bytes for Global Data at: 33db4f40

New Stack Pointer is: 33db4f30

RAM Configuration:

Bank #0: 30000000 64 MiB

relocation Offset is: 01fc5000

WARNING: Caches not enabled

monitor flash len: 0002939C

Now running in RAM - U-Boot at: 33fc5000

Flash: fwc addr 00000000 cmd f0 00f0 16bit x 16 bit

fwc addr 0000aaaa cmd aa 00aa 16bit x 16 bit

fwc addr 00005554 cmd 55 0055 16bit x 16 bit

fwc addr 0000aaaa cmd 90 0090 16bit x 16 bit

fwc addr 00000000 cmd f0 00f0 16bit x 16 bit

JEDEC PROBE: ID 1c 2249 0

fwc addr 00000000 cmd ff 00ff 16bit x 16 bit

fwc addr 00000000 cmd 90 0090 16bit x 16 bit

fwc addr 00000000 cmd ff 00ff 16bit x 16 bit

JEDEC PROBE: ID 16 ea00 0

*** failed ***

ERROR ### Please RESET the board

三、 移植 NOR FLASH

第二步 Flash: *** failed ***

ERROR ### Please RESET the board

卡在这里不动了

搜索“Flash:”

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "Flash:" * -nR

.....

arch/arm/lib/board.c:557: puts("Flash: ");

.....

进入查看

```
 puts("Flash: ");
 flash_size = flash_init();
 if (flash_size > 0) {
# ifdef CONFIG_SYS_FLASH_CHECKSUM
      print_size(flash_size, "");
      /*
       * Compute and print flash CRC if flashchecksum is set to 'y'
       *
       * NOTE: Maybe we should add some WATCHDOG_RESET()? XXX
       */
      if (getenv_yesno("flashchecksum") == 1) {
          printf("    CRC: %08X", crc32(0,
              (const unsigned char *) CONFIG_SYS_FLASH_BASE,
```

```
                flash_size));
        }
        putc('\n');
# else    /* !CONFIG_SYS_FLASH_CHECKSUM */
        print_size(flash_size, "\n");
# endif /* CONFIG_SYS_FLASH_CHECKSUM */
  } else {
        puts(failed);
        hang();
  }
```

failed 在该文件中的定义如下

```
static char *failed = "*** failed ***\n";
```

函数 hang()在该文件中的定义如下

```
void hang(void)
{
  puts("### ERROR ### Please RESET the board ###\n");
  for (;;);
}
```

说明是 flash 初始化失败

进入 drivers/mtd/cfi_flash.c：flash_init 函数

```
if (!flash_detect_legacy(cfi_flash_bank_addr(i), i))
```

flash_detect_legacy 函数去探测 flash

进入 flash_detect_legacy 函数

```
flash_read_jedec_ids(info);
debug("JEDEC PROBE: ID %x %x %x\n",
```

```
                    info->manufacturer_id,
                    info->device_id,
                    info->device_id2);
if (jedec_flash_match(info, info->start[0]))
    break;
else
    unmap_physmem((void *)info->start[0], MAP_NOCACHE);
```

jedec_flash_match 函数将读取到的 flash 信息与 jedec_table 进行匹配，如果匹配成功则填充 flash_info，否则返回 1，看来这里没有匹配成功，在 drivers/mtd/cfi_flash.c：flash_detect_legacy 函数去探测开发板的 flash，打开调试信息的宏，将探测到的 flash 信息打印出来

#define DEBUG　(也可以加载 tq2440.h 中)

重新编译，开发板从 NOR FLASH 启动，

```
EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 0a:1b:2c:3d:4e:5f
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
T #############
```

done

Bytes transferred = 168860 (2939c hex)

Starting application at 0x32000000 ...

U-Boot 2014.04 (Jun 29 2014 - 02:57:57)

U-Boot code: 32000000 -> 320246BC BSS: -> 3202A0A4

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

monitor len: 0002A0A4

ramsize: 04000000

TLB table from 33ff0000 to 33ff4000

Top of RAM usable for U-Boot at: 33ff0000

Reserving 168k for U-Boot at: 33fc5000

Reserving 2112k for malloc() at: 33db5000

Reserving 32 Bytes for Board Info at: 33db4fe0

Reserving 160 Bytes for Global Data at: 33db4f40

New Stack Pointer is: 33db4f30

RAM Configuration:

Bank #0: 30000000 64 MiB

relocation Offset is: 01fc5000

WARNING: Caches not enabled

monitor flash len: 0002939C

Now running in RAM - U-Boot at: 33fc5000

Flash: fwc addr 00000000 cmd f0 00f0 16bit x 16 bit

fwc addr 0000aaaa cmd aa 00aa 16bit x 16 bit

fwc addr 00005554 cmd 55 0055 16bit x 16 bit

fwc addr 0000aaaa cmd 90 0090 16bit x 16 bit

fwc addr 00000000 cmd f0 00f0 16bit x 16 bit

JEDEC PROBE: ID 1c 2249 0

fwc addr 00000000 cmd ff 00ff 16bit x 16 bit

fwc addr 00000000 cmd 90 0090 16bit x 16 bit

fwc addr 00000000 cmd ff 00ff 16bit x 16 bit

JEDEC PROBE: ID 16 ea00 0

*** failed ***

ERROR ### Please RESET the board

根据 debug("JEDEC PROBE: ID %x %x %x\n",

info->manufacturer_id,

info->device_id,

info->device_id2);

可以知道已经探测到开发板的 flash 的厂家 ID 为 0x1c，设备 ID 为 0x2249

在 jedec_table 表中增加 TQ2440 开发板的 NOR FLASH(EN29LV160AB)内容

```
#ifdef CONFIG_SYS_FLASH_LEGACY_1024Kx16
    {       /* TQ2440 EN29LV160AB */
            .mfr_id         = 0x1c,         /* manufacturer_id */
            .dev_id         = 0x2249,       /* device_id */
            .name           = "EON EN29LV160AB",
```

```
                .uaddr          = {/*  因为 NOR FLASH 的 ADDR0 接到了 S3C2440 的 ADDR1 */
                        [1] = MTD_UADDR_0x0555_0x02AA /* x16 */
                },
                .DevSize        = SIZE_2MiB,
                .CmdSet         = P_ID_AMD_STD,
                .NumEraseRegions= 4,
                .regions        = {
                        ERASEINFO(0x04000, 1),
                        ERASEINFO(0x02000, 2),
                        ERASEINFO(0x08000, 1),
                        ERASEINFO(0x10000, 31),
                }
        },
#endif
```

注释掉刚才打开的宏

drivers/mtd/cfi_flash.c

```
//#define DEBUG
```

并在 tq2440.h 中定义 CONFIG_SYS_FLASH_LEGACY_1024Kx16

```
//#define CONFIG_SYS_FLASH_LEGACY_512Kx16
#define CONFIG_SYS_FLASH_LEGACY_1024Kx16
```

重新编译 u-boot，从 NOR FLASH 启动开发板

```
EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 0a:1b:2c:3d:4e:5f
```

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x32000000

Loading: checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

T ############

done

Bytes transferred = 153568 (257e0 hex)

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 29 2014 - 03:10:56)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: ERROR: too many flash sectors

2 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: No ethernet found.

TQ2440 #

出现一个错误 too many flash sectors

搜索 too many flash sectors

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "too many flash sectors" * -nR

./drivers/mtd/jedec_flash.c:444: printf("ERROR: too many flash sectors\n");

查看代码

```
if (sect_cnt >= CONFIG_SYS_MAX_FLASH_SECT) {
        printf("ERROR: too many flash sectors\n");
        break;
    }
```

说明 CONFIG_SYS_MAX_FLASH_SECT 的值太小了，在配置文件 tq2440.h 修改该宏

#define CONFIG_SYS_MAX_FLASH_SECT (35) //根据 EN29LV160AB 芯片手册

重新编译，从 NOR FLASH 启动开发板

EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000

dm9000 i/o: 0x20000300, id: 0x90000a46

MAC: 0a:1b:2c:3d:4e:5f

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x32000000

Loading: checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

T ###########

done

Bytes transferred = 153568 (257e0 hex)

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 29 2014 - 03:12:36)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: No ethernet found.

TQ2440 #

TQ2440 # flinfo

Bank # 1: EON EN29LV160AB flash (16 x 16) Size: 2 MB in 35 Sectors

  AMD Legacy command set, Manufacturer ID: 0x1C, Device ID: 0x2249

  Erase timeout: 30000 ms, write timeout: 100 ms

  Sector Start Addresses:

  00000000 RO 00004000 RO 00006000 RO 00008000 RO 00010000 RO

  00020000 RO 00030000 00040000 00050000 00060000

  00070000 RO 00080000 00090000 000A0000 000B0000

  000C0000 000D0000 000E0000 000F0000 00100000

  00110000 00120000 00130000 00140000 00150000

  00160000 00170000 00180000 00190000 001A0000

  001B0000 001C0000 001D0000 001E0000 001F0000

测试 flash 读写是否正常

读取 0x32000000 地址的 0x10 字节数据到 0x0 地址，然后比较两份数据是否相等

注意：从 flinfo 中可以看出 0 地址所在扇区是 RO，要先解保护

TQ2440 # protect off all

Un-Protect Flash Bank # 1

TQ2440 # cp.b 32000000 0 10

Copy to Flash... done

TQ2440 # cmp.b 0 32000000 10

Total of 16 byte(s) were the same

再来读

TQ2440 # md.b 32000000 10

32000000: 13 00 00 ea 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5 ....

TQ2440 # md.b 0 10

00000000: 13 00 00 ea 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5 ....

读出的两份数据一样

至此 NOR FLASH 移植完毕

四、 移植网卡 DM9000

网卡 DM9000 的驱动为 drivers/net/dm9000x.c，我们需要将它编译进 u-boot，查看 drivers/net/Makefile

38 COBJS-$(CONFIG_DRIVER_DM9000) += dm9000x.o

如果定义了 CONFIG_DRIVER_DM9000 就将 dm9000x.o 编译进 u-boot，在配置文件 tq2440.h 中定义该宏

#define CONFIG_DRIVER_DM9000

在第一步已经经网卡 CS8900 的相关配置注释掉了

重新编译出错

dm9000x.c: In function 'dm9000_outblk_8bit':

dm9000x.c:156: error: 'DM9000_DATA' undeclared (first use in this function)

DM9000_DATA 没有定义，参考 mini2440.h 的配置如下

#define CONFIG_DRIVER_DM9000

#define CONFIG_DM9000_NO_SROM

（如果不设置这个宏，uboot 会打印类似：

Warning: dm9000 MAC addresses don't match:

Address in SROM is ff:ff:ff:ff:ff:ff

Address in environment is 00:0c:29:2a:5c:a5

的信息）

#define CONFIG_DM9000_BASE 0x20000000 //tq2440 开发板的网卡 dm9000 接在 S3C2440 的 bank4

#define DM9000_IO CONFIG_DM9000_BASE

#define DM9000_DATA (CONFIG_DM9000_BASE+4) // tq2440 开发板的网卡 dm9000 的 cmd 引脚接在 S3C2440 的 ADDR2

打开之前第一步暂时注释掉的宏

#define CONFIG_CMD_PING

重新编译，从 NOR FLASH 启动 u-boot

EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000

dm9000 i/o: 0x20000300, id: 0x90000a46

MAC: 0a:1b:2c:3d:4e:5f

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x32000000

Loading: checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

checksum bad

T ############

done

Bytes transferred = 154592 (25be0 hex)

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 29 2014 - 03:16:30)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: No ethernet found.

TQ2440 #

没有找到网路

搜索 Net：

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "Net:" * -nR

.....

arch/arm/lib/board.c:662: puts("Net: ");

.....

查看单板文件 board.c

puts("Net: ");

 eth_initialize(gd->bd);

进入 net/eth.c: eth_initialize 函数

if (board_eth_init(bis) < 0)

进入 board/tq2440/tq2440.c: board_eth_init 函数

修改如下

```
int board_eth_init(bd_t *bis)
{
 int rc = 0;
#ifdef CONFIG_CS8900
 rc = cs8900_initialize(0, CONFIG_CS8900_BASE);
#endif
#ifdef CONFIG_DRIVER_DM9000
 rc = dm9000_initialize(bis);
#endif
 return rc;
}
```

重新编译，从 NOR FLASH 启动 u-boot

```
EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 0a:1b:2c:3d:4e:5f
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
```

T ###########

done

Bytes transferred = 160268 (2720c hex)

Starting application at 0x32000000 ...¡ ø

U-Boot 2014.04 (Jun 29 2014 - 03:20:16)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: dm9000

TQ2440 # set ethaddr 00:12:34:56:ab:cd 设置 mac 地址

TQ2440 # set ipaddr 192.168.1.6 设置开发板 IP 地址

TQ2440 # set serverip 192.168.1.8 设置 tftp 服务器 IP 地址

TQ2440 # ping 192.168.1.8 ping 主机

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: 00:12:34:56:ab:cd

could not establish link

Using dm9000 device

host 192.168.1.8 is alive

TQ2440 # tftpboot 31000000 u-boot.bin　　下载文件

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: 00:12:34:56:ab:cd

could not establish link

Using dm9000 device

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x31000000

Loading: ###############

　　1.8 MiB/s

done

Bytes transferred = 198364 (306dc hex)

注意到新版 u-boot 在用 tftp 下载时会打印出下载速度，另外上面出现了一个错误 could not establish link

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "could not establish link" * -nR

./drivers/net/dm9000x.c:376:　　printf("could not establish link\n");

查看代码并修改

```
#if 0
 i = 0;
 while (!(dm9000_phy_read(1) & 0x20)) {     /* autonegation complete bit */
```

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

重新编译，从 NOR FLASH 重启开发板

```
EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 0a:1b:2c:3d:4e:5f
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
T ############
done
Bytes transferred = 160140 (2718c hex)
## Starting application at 0x32000000 ...¡ ø
```

U-Boot 2014.04 (Jun 29 2014 - 03:25:30)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: dm9000

现在重新下载刚才的 u-boot.bin

TQ2440 # tftp 0x30000000 u-boot.bin

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: ff:ff:ff:ff:ff:ff

WARNING: Bad MAC address (uninitialized EEPROM?)

operating at 100M full duplex mode

Using dm9000 device

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'u-boot.bin'.

Load address: 0x30000000

Loading: T ############

30.3 KiB/s

done

Bytes transferred = 160140 (2718c hex)

刚才的错误解决了

至此网卡 DM9000 移植完毕

五、 支持 NAND FLASH 读写

修改配置文件 include/configs/tq2440.h 打开之前注释掉的 NAND 相关的宏

#define CONFIG_CMD_NAND

编译出错

s3c2410_nand.c: In function 's3c2410_hwcontrol':

s3c2410_nand.c:57: warning: implicit declaration of function 's3c2410_get_base_nand'

s3c2410_nand.c:57: warning: initialization makes pointer from integer without a cast

s3c2410_nand.c:72: error: dereferencing pointer to incomplete type

s3c2410_nand.c:72: error: dereferencing pointer to incomplete type

s3c2410_nand.c:75: error: dereferencing pointer to incomplete type

s3c2410_nand.c:75: error: dereferencing pointer to incomplete type

查看代码

struct s3c2410_nand *nand = s3c2410_get_base_nand();

struct s3c2410_nand 的定义

#ifdef CONFIG_S3C2410

/* NAND FLASH (see S3C2410 manual chapter 6) */

```
struct s3c2410_nand {
 u32      nfconf;
 u32      nfcmd;
 u32      nfaddr;
 u32      nfdata;
 u32      nfstat;
 u32      nfecc;
};
#endif
```

s3c2410_get_base_nand()在 s3c2410.h 中声明

s3c2410_nand.c 中包含了头文件 s3c24x0_cpu.h，该文件内容如下

```
#ifdef CONFIG_S3C2400
 #include <asm/arch/s3c2400.h>
#elif defined CONFIG_S3C2410
 #include <asm/arch/s3c2410.h>
#elif defined CONFIG_S3C2440
 #include <asm/arch/s3c2440.h>
#else
 #error Please define the s3c24x0 cpu type
#endif
```

而我在配置文件 tq2440 中注释了宏 CONFIG_S3C2410，添加了 CONFIG_S3C2440

查看 drivers/mtd/nand/Makefile

75 COBJS-$(CONFIG_NAND_S3C2410) += s3c2410_nand.o

CONFIG_NAND_S3C2410 在 tq2440.h 中定义

将 drivers/mtd/nand/s3c2410_nand.c 拷贝为 drivers/mtd/nand/s3c2440_nand.c

pengdl@debian:~/work/tq2440/u-boot-2014.04$ cp drivers/mtd/nand/s3c2410_nand.c drivers/mtd/nand/s3c2440_nand.c

在 drivers/mtd/nand/Makefile 文件中增加一行

```
56      obj-$(CONFIG_NAND_S3C2440) += s3c2440_nand.o
```

将 s3c2440_nand.c 中所有的 2410 替换为 2440

修改配置文件 include/configs/tq2440.h

```
187 /*
188    * NAND configuration
189    */
190 #ifdef CONFIG_CMD_NAND
191 #ifdef CONFIG_S3C2440
192 #define CONFIG_NAND_S3C2440
193 #define CONFIG_SYS_S3C2440_NAND_HWECC
194 #else
195 #define CONFIG_NAND_S3C2410
196 #define CONFIG_SYS_S3C2410_NAND_HWECC
197 #endif
198 #define CONFIG_SYS_MAX_NAND_DEVICE    1
199 #define CONFIG_SYS_NAND_BASE              0x4E000000
200 #endif
```

重新编译，从 NOR FLASH 启动开发板

```
EmbedSky> tftp 0x32000000 u-boot.bin;go 0x32000000
```

.....

U-Boot 2014.04 (Jun 29 2014 - 03:42:02)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

NAND: 0 MiB

*** Warning - bad CRC, using default environment

In: serial

Out: serial

Err: serial

Net: dm9000

TQ2440 #

搜索 NAND:

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "NAND:" * -nR

.....

arch/arm/lib/board.c:584: puts("NAND: ");

.....

查看代码

#if defined(CONFIG_CMD_NAND)

 puts("NAND: ");

```
 nand_init();          /* go init the NAND */
#endif
```

一路跟踪下去，列出函数调用顺序

```
nand_init-> nand_init_chip-> board_nand_init
                                -> nand_scan-> nand_scan_ident
```

其中 board_nand_init 是与板相关的初始化

nand_scan_ident 函数中

```
 /* Read the flash type */
 type = nand_get_flash_type(mtd, chip, busw,
                &nand_maf_id, &nand_dev_id, table);

 if (IS_ERR(type)) {
#ifndef CONFIG_SYS_NAND_QUIET_TEST
      printk(KERN_WARNING "No NAND device found!!!\n");
#endif
      chip->select_chip(mtd, -1);
      return PTR_ERR(type);
 }
```

说明上面的 nand_get_flash_type 获取失败

修改 drivers/mtd/nand/s3c2440_nand.c 中的 board_nand_init()函数

```
#define S3C2440_NFCONF_TACLS(x)      ((x)<<12)
#define S3C2440_NFCONF_TWRPH0(x)     ((x)<<8)
#define S3C2440_NFCONF_TWRPH1(x)     ((x)<<4)
.....
```

```
#if defined(CONFIG_S3C24XX_CUSTOM_NAND_TIMING)
  tacls    = CONFIG_S3C24XX_TACLS;
  twrph0 = CONFIG_S3C24XX_TWRPH0;
  twrph1 =    CONFIG_S3C24XX_TWRPH1;
#else
  tacls = 0;
  twrph0 = 1;
  twrph1 = 0;
#endif


  //cfg = S3C2440_NFCONF_EN;
  cfg |= S3C2440_NFCONF_TACLS(tacls);
  cfg |= S3C2440_NFCONF_TWRPH0(twrph0);
  cfg |= S3C2440_NFCONF_TWRPH1(twrph1);
  writel(cfg, &nand_reg->nfconf);
/*初始化 ECC、禁止片选、使能 NAND FLASH 控制器*/
  writel((1 << 4)|(1 << 1)|(1 << 0), &nand_reg->nfcont);
```

修改默认的 drivers/mtd/nand/nand_base.c: nand_select_chip 函数

```
  case 0: //  选中
        chip->cmd_ctrl(mtd, NAND_CMD_NONE, NAND_CTRL_CLE | NAND_CTRL_CHANGE);
        break;
```

修改 s3c2440_hwcontrol()函数

```
  if (ctrl & NAND_CTRL_CHANGE) {
        ulong IO_ADDR_W = (ulong)nand;
```

```
    if (!(ctrl & NAND_CLE))
        IO_ADDR_W |= S3C2440_ADDR_NCLE;
    if (!(ctrl & NAND_ALE))
        IO_ADDR_W |= S3C2440_ADDR_NALE;

    if (cmd == NAND_CMD_NONE)
        IO_ADDR_W = &nand->nfdata;

    chip->IO_ADDR_W = (void *)IO_ADDR_W;

    if (ctrl & NAND_NCE)    // 使能选中
        writel(readl(&nand->nfcont) & ~(1 << 1),
                &nand->nfcont);
    else          // 取消选中
        writel(readl(&nand->nfcont) | (1 << 1),
                &nand->nfcont);
}
```

修改

```
#define S3C2440_ADDR_NALE 8
#define S3C2440_ADDR_NCLE 0xc
#define S3C2440_NFCONF_nFCE             (1<<1)
```

重新编译烧到，从 NOR FLASh 启动开发板

```
TQ2440 # tftp 32000000 u-boot.bin;go 32000000
```

测试

将 SDRAM 的 0x32000000 地址的 0xff 字节数据写到 NAND FLASH 的 0 地址

TQ2440 # nand write 32000000 0 ff

NAND write: device 0 offset 0x0, size 0xff

255 bytes written: OK

从 NAND FLASH 的 0 地址读 0xff 字节数据到 SDRAM 的 0x31000000 地址

TQ2440 # nand read 31000000 0 ff

NAND read: device 0 offset 0x0, size 0xff

255 bytes read: OK

比较

TQ2440 # cmp.b 31000000 32000000 ff

Total of 255 bytes were the same

六、 修改代码支持 NOR FLASH 启动

修改配置文件 tq2440.h

#define CONFIG_SYS_TEXT_BASE 0x0

(这里必须设置为 0，因为这个版本的 uboot 中尚未进行代码重定位之前，就调用了很多全局性的代码，所以需要设置为 0，否则程序会跑飞)

//#define CONFIG_SKIP_LOWLEVEL_INIT

（如果不定义 CONFIG_SKIP_LOWLEVEL_INIT，u-boot 在从 NorFlash 启动时会执行内存初始化代码

```
190         /*
191          * we do sys-critical inits only at reboot,
192          * not when booting from ram!
193          */
194 #ifndef CONFIG_SKIP_LOWLEVEL_INIT
195         bl    cpu_init_crit
196 #endif
197
```

```
198         bl    _main
)
```

修改内存初始化代码 board/tq2440/lowlevel_init.S

```
#define REFCNT              0x4f4       #修改内存初始化参数
```

重新编译，从 NOR FLASH 启动，使用原先好的 u-boot 将新的 u-boot.bin 烧写到 NOR FLASH

```
EmbedSky> tftp 0x32000000 u-boot.bin
dm9000 i/o: 0x20000300, id: 0x90000a46
MAC: 0a:1b:2c:3d:4e:5f
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
checksum bad
T ###############
done
Bytes transferred = 215096 (34838 hex)
EmbedSky> protect off all
Un-Protect Flash Bank # 1
EmbedSky> erase 0 +40000
Erasing sector    0 ... ok.
Erasing sector    1 ... ok.
```

Erasing sector 2 ... ok.

Erasing sector 3 ... ok.

Erasing sector 4 ... ok.

Erasing sector 5 ... ok.

Erasing sector 6 ... ok.

Erased 7 sectors

EmbedSky> cp.b 0x32000000 0x0 0x40000

Copy to Flash... done

ÿmbedSky> reset

U-Boot 2014.04 (Jun 29 2014 - 04:07:20)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 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: dm9000

TQ2440 #

七、 保存环境变量到 NAND FLASH 并添加分区

默认是把环境变量保存到 NOR FLASH 的，环境变量是通过执行 saveenv 命令保存的，查看一下该命令的实现代码。搜索“saveenv”

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "saveenv" common/ -nR

common/env_flash.c:122:int saveenv(void)

common/env_flash.c:245:int saveenv(void)

common/env_nand.c:174:int saveenv(void)

common/env_nand.c:226:int saveenv(void)

搜索出了 env_flash.c 和 env_fnand.c，分别用于将环境变量保存到 NOR FLASH 和 NAND FLASH

查看 common/Makefile

57 COBJS-$(CONFIG_ENV_IS_IN_FLASH) += env_flash.o

60 COBJS-$(CONFIG_ENV_IS_IN_NAND) += env_nand.o

我们需要在配置文件 tq2440.h 中去掉 CONFIG_ENV_IS_IN_FLASH，增加

CONFIG_ENV_IS_IN_NAND

修改配置文件 tq2440.h

```
#if 0
#define CONFIG_ENV_ADDR              (CONFIG_SYS_FLASH_BASE + 0x070000)
#define CONFIG_ENV_IS_IN_FLASH
#define CONFIG_ENV_SIZE          0x10000
#endif
#define CONFIG_ENV_IS_IN_NAND
/* allow to overwrite serial and ethaddr */
#define CONFIG_ENV_OVERWRITE
```

重新编译，出错

/home/work/u-boot-2013.01/include/environment.h:88: error: #error "Need to define CONFIG_ENV_OFFSET when using CONFIG_ENV_IS_IN_NAND"

/home/work/u-boot-2013.01/include/environment.h:95: error: #error "Need to define CONFIG_ENV_SIZE when using CONFIG_ENV_IS_IN_NAND"

在配置文件 tq2440.h 中定义这两个宏

#define CONFIG_ENV_IS_IN_NAND

#define CONFIG_ENV_OFFSET 0x40000 // 256K for u-boot

#define CONFIG_ENV_SIZE 0x20000 // 128K for environment

重新编译，从 NOR FLASH 启动开发板

U-Boot 2014.04 (Jun 29 2014 - 04:31:22)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 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:      dm9000
TQ2440 # protect off all;erase 0 +40000;tftp 32000000 u-boot.bin;cp.b 32000000 0 40000
Un-Protect Flash Bank # 1


..... done
Erased 7 sectors
dm9000 i/o: 0x20000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: ff:ff:ff:ff:ff:ff
WARNING: Bad MAC address (uninitialized EEPROM?)
operating at 100M full duplex mode
Using dm9000 device
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: T ###############
                41 KiB/s
done
Bytes transferred = 215572 (34a14 hex)
Copy to Flash... 9....8....7....6....5....4....3....2....1....done
TQ2440 # reset
resetting ...


U-Boot 2014.04 (Jun 29 2014 - 04:31:22)
```

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 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: dm9000

出现*** Warning - bad CRC, using default environment，这是因为 NAND FLASH 中没有有效的环境变量，执行 saveenv 命令将环境变量保存到 NAND FLASH，重启就不会有这样的提示了

TQ2440 # saveenv

Saving Environment to NAND...

Erasing NAND...

Erasing at 0x40000 -- 100% complete.

Writing to NAND... OK

TQ2440 # reset

resetting ...

U-Boot 2014.04 (Jun 29 2014 - 04:31:22)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: 2 MiB

NAND: 256 MiB

In: serial

Out: serial

Err: serial

Net: dm9000

TQ2440 # mtd

Unknown command 'mtd' - try 'help'

默认没有分区命令，查看一下 common/Makefile

140 COBJS-$(CONFIG_CMD_MTDPARTS) += cmd_mtdparts.o

说明需要定义 CONFIG_CMD_MTDPARTS 才会编译 mtd 命令

在 tq2440.h 中其实已经定义了该宏，只是之前被我注释掉了，现在打开它

#if 0

#define CONFIG_CMD_FAT

#define CONFIG_CMD_EXT2

#define CONFIG_CMD_UBI

#define CONFIG_CMD_UBIFS

#endif

#define CONFIG_CMD_MTDPARTS

#if 0

#define CONFIG_MTD_DEVICE

#define CONFIG_MTD_PARTITIONS

#define CONFIG_YAFFS2

#define CONFIG_RBTREE

#endif

重新编译，出错

/home/work/u-boot-2013.01/common/cmd_mtdparts.c:306: undefined reference to `get_mtd_device_nm'

查看代码知道 get_mtd_device_nm 在 drivers/mtd/mtdcore.c 中定义，查看 drivers/mtd/Makefile

28 COBJS-$(CONFIG_MTD_DEVICE) += mtdcore.o

需要定义 CONFIG_MTD_DEVICE 才会编译 mtdcore.c

在 tq2440.h 中其实已经定义了该宏，只是之前被我注释掉了，现在打开它

#if 0

#define CONFIG_CMD_FAT

#define CONFIG_CMD_EXT2

#define CONFIG_CMD_UBI

#define CONFIG_CMD_UBIFS

#endif

#define CONFIG_CMD_MTDPARTS

#define CONFIG_MTD_DEVICE

#if 0

#define CONFIG_MTD_PARTITIONS

#define CONFIG_YAFFS2

#define CONFIG_RBTREE

#endif

重新编译，从 NOR FLASH 启动开发板

TQ2440 # protect off all;erase 0 +40000;tftp 32000000 u-boot.bin;cp.b 32000000 0 40000

……

TQ2440 # reset

……

TQ2440 # mtdparts

mtdids not defined, no default present

出错了，搜索 mtdids not defined, no default present

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "mtdids not defined, no default present" * -nR

common/cmd_mtdparts.c:1751:	printf("mtdids not defined, no default present\n");

查看代码

```
if (mtdids_default) {
        debug("mtdids variable not defined, using default\n");
        ids = mtdids_default;
        setenv("mtdids", (char *)ids);
    } else {
        printf("mtdids not defined, no default present\n");
        return 1;
    }
```

说明 mtdids_default 为假，mtdids_default 定义为

#if defined(MTDIDS_DEFAULT)

```
static const char *const mtdids_default = MTDIDS_DEFAULT;
#else
static const char *const mtdids_default = NULL;
#endif
#if defined(MTDPARTS_DEFAULT)
static const char *const mtdparts_default = MTDPARTS_DEFAULT;
#else
static const char *const mtdparts_default = NULL;
#endif
```

需要在配置文件 tq2440.h 定义 MTDIDS_DEFAULT 和 MTDPARTS_DEFAULT，参考 common/cmd_mtdparts.c

```
* Examples:
 *
 * 1 NOR Flash, with 1 single writable partition:
 * mtdids=nor0=edb7312-nor
 * mtdparts=mtdparts=edb7312-nor:-
 *
 * 1 NOR Flash with 2 partitions, 1 NAND with one
 * mtdids=nor0=edb7312-nor,nand0=edb7312-nand
 * mtdparts=mtdparts=edb7312-nor:256k(ARMboot)ro,-(root);edb7312-nand:-(home)
```

在 tq2440.h 中定义

```
#define MTDIDS_DEFAULT  "nand0=tq2440-0"
#define MTDPARTS_DEFAULT   "mtdparts=tq2440-0:1m(u-boot),"     \
                        "1m(params),"      \
                        "3m(kernel)," \
```

"-(rootfs)"

在 tq2440.h 中设置默认环境变量

#define CONFIG_NETMASK 255.255.255.0

#define CONFIG_IPADDR 192.168.1.6

#define CONFIG_SERVERIP 192.168.1.8

#define CONFIG_ETHADDR 00:11:22:33:44:aa

#define CONFIG_BOOTARGS "root=/dev/mtdblock2 rootfstype=yaffs2 init=/linuxrc console=ttySAC0,115200"

#define CONFIG_BOOTCOMMAND "nand read 32000000 kernel\;bootm 32000000"

重新编译，从 NOR FLASH 启动开发板

TQ2440 # protect off all;erase 0 +40000;tftp 32000000 u-boot.bin;cp.b 32000000 0 40000

……

TQ2440 # reset

……

TQ2440 # mtdparts

mtdparts variable not set, see 'help mtdparts'

no partitions defined

defaults:

mtdids : nand0=tq2440-0

mtdparts: mtdparts=tq2440-0:1m(u-boot),1m(params),3m(kernel),-(rootfs)

TQ2440 # mtdparts default

TQ2440 # save 注意一定要保存，否则重启就没了

TQ2440 # mtdparts

device nand0 <tq2440-0>, # parts = 4

#: name	size	offset	mask_flags
0: u-boot	0x00100000	0x00000000	0
1: params	0x00100000	0x00100000	0
2: kernel	0x00300000	0x00200000	0
3: rootfs	0x0fb00000	0x00500000	0

active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00000000

defaults:

mtdids : nand0=tq2440-0

mtdparts: mtdparts=tq2440-0:1m(u-boot),1m(params),3m(kernel),-(rootfs)

TQ2440 #

八、 支持 NAND FLASH 启动

新版 u-boot 在链接时加了“-pie”选项

-pie

Produce a position independent executable on targets which support it. For predictable results, you must also specify the same set of options that were used to generate code (-fpie, -fPIE, or model suboptions) when you specify this option.

产生的代码中，没有绝对地址，全部使用相对地址，故而代码可以被加载器加载到内存的任意位置，都可以正确的执行。

最终 u-boot.bin 中多了这些段

```
.rel.dyn : {
    __rel_dyn_start = .;
    *(.rel*)
    __rel_dyn_end = .;
```

```
}

.dynsym : {
    __dynsym_start = .;
    *(.dynsym)
}
```

从 NOR FLASH 把代码复制到 SDRAM，程序的链接地址是 0，访问全局变量、静态变量、调用函数时是使用基于 0 地址编译得到的地址，现在把程序复制到了 SDRAM(0x3000000),需要修改代码，把原来的地址改为新地址。这样太复杂了，还是使用老版本的方法。

去掉“-pie”选项，在 u-boot 源码搜索“-pie”

```
pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "\-pie" . -nR
.....
./arch/arm/config.mk:83:LDFLAGS_u-boot += -pie
.....
```

去除 arch/arm/config.mk:83:LDFLAGS_u-boot += -pie 中的“-pie”

```
 82 # needed for relocation
 83 #LDFLAGS_u-boot += -pie
```

修改配置文件 include/configs/tq2440.h，给 u-boot 分配 512KB

```
#define CONFIG_SYS_TEXT_BASE    0x33f80000
```

增加文件 board/tq2440/nand_read_ll.c 并修改相应的 Makefile

```
obj-y    := tq2440.o
obj-y    += nand_read_ll.o
obj-y    += lowlevel_init.o
```

nand_read_ll.c 文件内容如下：

```
/* NAND FLASH 控制器  */
```

```
#define NFCONF (*((volatile unsigned long *)0x4E000000))
#define NFCONT (*((volatile unsigned long *)0x4E000004))
#define NFCMMD (*((volatile unsigned char *)0x4E000008))
#define NFADDR (*((volatile unsigned char *)0x4E00000C))
#define NFDATA (*((volatile unsigned char *)0x4E000010))
#define NFSTAT (*((volatile unsigned char *)0x4E000020))

static void nand_read_ll(unsigned int addr, unsigned char *buf, unsigned int len);



static int isBootFromNorFlash(void)
{
  volatile int *p = (volatile int *)0;
  int val;

  val = *p;
  *p = 0x12345678;
  if (*p == 0x12345678)
  {
      /*  写成功, 是 nand 启动  */
      *p = val;
      return 0;
  }
  else
  {
```

```
        /* NOR 不能像内存一样写 */
        return 1;
    }
}

void nand_init_ll(void)
{
#define TACLS      0
#define TWRPH0     1
#define TWRPH1     0
    /* 设置时序 */
    NFCONF = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4);
    /* 使能 NAND Flash 控制器, 初始化 ECC, 禁止片选 */
    NFCONT = (1<<4)|(1<<1)|(1<<0);
}

void copy_code_to_sdram(unsigned char *src, unsigned char *dest, unsigned int len)
{
    int i = 0;

    /* 如果是 NOR 启动 */
    if (isBootFromNorFlash())
    {
        while (i < len)
        {
```

```
            dest[i] = src[i];
            i++;
        }
    }
    else
    {
        nand_init_ll();
        nand_read_ll((unsigned int)src, dest, len);
    }
}

void clear_bss(void)
{
    extern int __bss_start, __bss_end;
    int *p = &__bss_start;

    for (; p < &__bss_end; p++)
        *p = 0;
}



static void nand_select(void)
{
    NFCONT &= ~(1<<1);
}
```

```
static void nand_deselect(void)
{
  NFCONT |= (1<<1);
}

static void nand_cmd(unsigned char cmd)
{
  volatile int i;
  NFCMMD = cmd;
  for (i = 0; i < 10; i++);
}

static void nand_addr(unsigned int addr)
{
  unsigned int col  = addr % 2048;
  unsigned int page = addr / 2048;
  volatile int i;

  NFADDR = col & 0xff;
  for (i = 0; i < 10; i++);
  NFADDR = (col >> 8) & 0xff;
  for (i = 0; i < 10; i++);

  NFADDR    = page & 0xff;
```

```
  for (i = 0; i < 10; i++);
  NFADDR    = (page >> 8) & 0xff;
  for (i = 0; i < 10; i++);
  NFADDR    = (page >> 16) & 0xff;
  for (i = 0; i < 10; i++);
}

static void nand_wait_ready(void)
{
  while (!(NFSTAT & 1));
}

static unsigned char nand_data(void)
{
  return NFDATA;
}

static void nand_read_ll(unsigned int addr, unsigned char *buf, unsigned int len)
{
  int col = addr % 2048;
  int i = 0;

  /* 1. 选中 */
  nand_select();
```

```
while (i < len)
{
    /* 2. 发出读命令 00h */
    nand_cmd(0x00);

    /* 3. 发出地址(分 5 步发出) */
    nand_addr(addr);

    /* 4. 发出读命令 30h */
    nand_cmd(0x30);

    /* 5. 判断状态  */
    nand_wait_ready();

    /* 6. 读数据  */
    for (; (col < 2048) && (i < len); col++)
    {
        buf[i] = nand_data();
        i++;
        addr++;
    }

    col = 0;
}
```

```
 /* 7. 取消选中 */
 nand_deselect();
}
```

通过阅读 u-boot 源码 arch/arm/cpu/arm920t/start.S

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
 bl  cpu_init_crit
#endif

 bl  _main
```

跳转到 arch/arm/lib/crt0.S：_main

修改 arch/arm/lib/crt0.S 为:

```
ENTRY(_main)

/*
 * Set up initial C runtime environment and call board_init_f(0).
 */

#if defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_STACK)
 ldr sp, =(CONFIG_SPL_STACK)
#else
 ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
#endif
 bic sp, sp, #7  /* 8-byte alignment for ABI compliance */
 sub sp, sp, #GD_SIZE /* allocate one GD above SP */
```

```
  bic sp, sp, #7   /* 8-byte alignment for ABI compliance */
  mov     r9, sp         /* GD is above SP */
#if 1
__TEXT_BASE:
  .word   CONFIG_SYS_TEXT_BASE

  mov r0, #0
  ldr r1, __TEXT_BASE

  ldr  r2, __TEXT_BASE
  ldr  r3, =__bss_end

  sub r2, r3, r2

  bl copy_code_to_sdram
  bl clear_bss

  ldr pc, =call_board_init_f

call_board_init_f:
  mov     r0, #0
  bl   board_init_f

  ldr  sp, [r9, #GD_START_ADDR_SP]     /* sp = gd->start_addr_sp */
  bic sp, sp, #7              /* 8-byte alignment for ABI compliance */
```

```
  ldr  r9, [r9, #GD_BD]       /* r9 = gd->bd */
  sub r9, r9, #GD_SIZE       /* new GD is below bd */

  ldr r1, __TEXT_BASE
  bl    board_init_r
#else
  mov     r0, #0
  bl   board_init_f

#if ! defined(CONFIG_SPL_BUILD)

/*
  * Set up intermediate environment (new sp and gd) and call
  * relocate_code(addr_moni). Trick here is that we'll return
  * 'here' but relocated.
  */

  ldr  sp, [r9, #GD_START_ADDR_SP]     /* sp = gd->start_addr_sp */
  bic sp, sp, #7   /* 8-byte alignment for ABI compliance */
  ldr  r9, [r9, #GD_BD]       /* r9 = gd->bd */
  sub r9, r9, #GD_SIZE       /* new GD is below bd */

  adr lr, here
  ldr  r0, [r9, #GD_RELOC_OFF]        /* r0 = gd->reloc_off */
  add lr, lr, r0
```

```
  ldr  r0, [r9, #GD_RELOCADDR]      /* r0 = gd->relocaddr */
  b    relocate_code
here:

/* Set up final (full) environment */

  bl   c_runtime_cpu_setup  /* we still call old routine here */

  ldr  r0, =__bss_start     /* this is auto-relocated! */
  ldr  r1, =__bss_end           /* this is auto-relocated! */

  mov     r2, #0x00000000         /* prepare zero to clear BSS */

clbss_l:cmp r0, r1                /* while not at end of BSS */
  strlo     r2, [r0]       /* clear 32-bit BSS word */
  addlo   r0, r0, #4          /* move to next */
  blo clbss_l

  bl coloured_LED_init
  bl red_led_on

  /* call board_init_r(gd_t *id, ulong dest_addr) */
  mov          r0, r9                              /* gd_t */
  ldr  r1, [r9, #GD_RELOCADDR] /* dest_addr */
  /* call board_init_r */
```

```
 ldr pc, =board_init_r  /* this is auto-relocated! */
 /* we should not return here. */
#endif
#endif

ENDPROC(_main)
```

修改 arch/arm/lib/board.c 中的函数 board_init_f 为：

```
00262:
00263: unsigned int board_init_f(ulong bootflag)
00264: {
00265:         bd_t *bd;
00266:         init_fnc_t **init_fnc_ptr;
```

然后再 board_init_f 的结尾添加：

```
00461:          }
00462:          memcpy(id, (void *)gd, sizeof(gd_t));
00463:
00464:          return (unsigned int)id;
00465: } ? end board_init_f ?
```

即将 board_init_r 要用的第一个参数返回，也就是重定向后 gd 结构体的起始地址。

修改 include/common.h：

```
00297: /* arch/$(ARCH)/lib/board.c */
00298: unsigned int    board_init_f(ulong);
00299: void    board_init_r  (gd_t *, ulong)
```

修改 arch/arm/lib/board.c 中的函数 board_init_f

```
//addr -= gd->mon_len;
//addr &= ~(4096 - 1);
addr = CONFIG_SYS_TEXT_BASE;
```

然后编译，编译完成后，执行 make u-boot.dis，生成 u-boot 的反汇编文件，看一下 start.o、nand_read_ll.o、lowlevel_init.o 是否被连接到了 u-boot 的前面 4KB 范围内：

```
33f807d8:       eb000321        bl      33f81464 <copy_code_to_sdram>
33f807dc:       eb000309        bl      33f81408 <clear_bss>
33f807e0:       e59ff024        ldr     pc, [pc, #36]   ; 33f8080c <call_board_init_f+0x28>
```

从上面的图中可以看到，copy_code_to_sdram 和 clear_bss 的地址已经超过 4KB 了，前面我们把 CONFIG_SYS_TEXT_BASE 设置为了 0x33f80000，但是 copy_code_to_sdram 的地址已经到了 0x33f81464，0x33f81464-0x33f80000=0x1464>0x1000，显然超多了 4KB（十六进制就是 0x1000），所以需要修改链接脚本和 Makefile，将 nand_read_ll 和 lowlevel_init.S 链接到 uboot 的前 4KB 内，具体做法如下：

修改链接脚本 arch/arm/cpu/u-boot.lds 把 start.o、nand_read_ll.o、lowlevel_init.o 编译到前面 4KB

修改 board/tq2440/Makefile：

```
obj-y     := tq2440.o

extra-y  := nand_read_ll.o

extra-y  += lowlevel_init.o
```

修改 arch/arm/cpu/u-boot.lds：

```
 . = ALIGN(4);

 .text :

 {

      *(.__image_copy_start)

      CPUDIR/start.o (.text*)

      board/tq2440/lowlevel_init.o (.text*)

      board/tq2440/nand_read_ll.o (.text*)

      *(.text*)

 }
```

重新编译：

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

```
  HOSTLD    tools/dumpimage
  HOSTLD    tools/mkimage
  CC        arch/arm/lib/board.o
  LD        arch/arm/lib/built-in.o
  AS        board/tq2440/lowlevel_init.o
  CC        common/main.o
  CC        common/cmd_version.o
  LD        common/built-in.o
  CC        lib/display_options.o
  LD        lib/built-in.o
  LDS       u-boot.lds
  LD        u-boot
u-boot contains unexpected relocations:
make: *** [checkarmreloc] Error 1
```

搜索“u-boot contains unexpected relocations”

```
pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "u-boot contains unexpected relocations" * -nR
```

没有搜索到任何内容，那么我们搜索 checkarmreloc：

```
pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "checkarmreloc" * -nR
arch/arm/config.mk:105:ALL-y += checkarmreloc
Makefile:1123:checkarmreloc: u-boot
```

在顶层 Makefile 中有如下语句:

```
1121 # ARM relocations should all be R_ARM_RELATIVE (32-bit) or
1122 # R_AARCH64_RELATIVE (64-bit).
1123 checkarmreloc: u-boot
1124        @RELOC="`$(CROSS_COMPILE)readelf -r -W $< | cut -d ' ' -f 4 | \
```

```
1125                grep R_A | sort -u`"; \
1126        if test "$$RELOC" != "R_ARM_RELATIVE" -a \
1127                "$$RELOC" != "R_AARCH64_RELATIVE"; then \
1128                echo "$< contains unexpected relocations: $$RELOC"; \
1129                false; \
1130        fi
```

那么我们就别编译 checkarmreloc 了，所以我们修改 arch/arm/config.mk 的第 105 行：

105 #ALL-y += checkarmreloc

重新编译，从 NOR FLASH 启动开发板，将 u-boot.bin 烧到 NAND FLASH

```
TQ2440 # nand erase 0 40000;tftp 32000000 u-boot.bin;nand write 32000000 0 40000

NAND erase: device 0 offset 0x0, size 0x40000
Erasing at 0x20000 -- 100% complete.
OK
dm9000 i/o: 0x20000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:0c:29:2a:5c:a5
operating at 100M full duplex mode
Using dm9000 device
TFTP from server 192.168.1.8; our IP address is 192.168.1.6
Filename 'u-boot.bin'.
Load address: 0x32000000
Loading: T ################
         39.1 KiB/s
done
```

Bytes transferred = 209400 (331f8 hex)

NAND write: device 0 offset 0x0, size 0x40000

262144 bytes written: OK

TQ2440 #

从 NAND FLASh 启动开发板

TQ2440 # reset

resetting ...

U-Boot 2014.04 (Jun 29 2014 - 07:58:40)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: *** failed ***

ERROR ### Please RESET the board

卡在这里不动了，由于从 NAND 启动，CPU 检测不到 NOR FLASH，具体代码如下 arch/arm/lib/board.c

```
flash_size = flash_init();
 if (flash_size > 0) {
# ifdef CONFIG_SYS_FLASH_CHECKSUM
        print_size(flash_size, "");
```

```
        /*
         * Compute and print flash CRC if flashchecksum is set to 'y'
         *
         * NOTE: Maybe we should add some WATCHDOG_RESET()? XXX
         */
        if (getenv_yesno("flashchecksum") == 1) {
            printf("    CRC: %08X", crc32(0,
                (const unsigned char *) CONFIG_SYS_FLASH_BASE,
                flash_size));
        }
        putc('\n');
# else    /* !CONFIG_SYS_FLASH_CHECKSUM */
        print_size(flash_size, "\n");
# endif /* CONFIG_SYS_FLASH_CHECKSUM */
  } else {
        puts(failed);
        hang();
  }
```

其中 failed 定义为

```
#if !defined(CONFIG_SYS_NO_FLASH)
static char *failed = "*** failed ***\n";
#endif
```

函数 hang()定义为

```
void hang(void)
{
```

```
 puts("### ERROR ### Please RESET the board ###\n");
 for (;;);
}
```

我们直接注释掉上面的 hang();

```
# endif /* CONFIG_SYS_FLASH_CHECKSUM */
 } else {
      puts(failed);
      //hang();
 }
#endif
```

重新编译，从 NOR FLASH 启动开发板，将 u-boot.bin 烧到 NAND FLASH，从 NAND 启动

```
CPUID: 32440001
FCLK:          400 MHz
HCLK:          100 MHz
PCLK:           50 MHz
DRAM:    64 MiB
WARNING: Caches not enabled
Flash: *** failed ***
NAND:    256 MiB
In:       serial
Out:      serial
Err:      serial
Net:      dm9000
Hit any key to stop autoboot:    0
TQ2440 #
```

九、 支持烧写 yaffs 文件系统

执行? Nand

TQ2440 # ? nand

nand - NAND sub-system

Usage:

……

没有.yaffs 后缀，在源码目录搜索.yaffs

pengdl@debian:~/work/tq2440/u-boot-2014.04$ grep "\.yaffs" * -nR

.....

common/cmd_nand.c:709: } else if (!strcmp(s, ".yaffs")) {

.....

查看代码

```
 708 #ifdef CONFIG_CMD_NAND_YAFFS
 709         } else if (!strcmp(s, ".yaffs")) {
 710             if (read) {
 711                 printf("Unknown nand command suffix '%s'.\n", s);
 712                 return 1;
 713             }
 714             ret = nand_write_skip_bad(nand, off, &rwsize, NULL,
 715                                       maxsize, (u_char *)addr,
 716                                       WITH_YAFFS_OOB);
 717 #endif
```

需要在配置文件 tq2440.h 中定义 CONFIG_CMD_NAND_YAFFS

#define CONFIG_CMD_NAND_YAFFS

重新编译，烧写到 NandFlash 中：

TQ2440 # nand ?

nand - NAND sub-system

Usage:

……

nand write.yaffs - addr off|partition size

write 'size' bytes starting at offset 'off' with yaffs format

from memory address 'addr', skipping bad blocks.

……

TQ2440 #

先直接烧写试一下

TQ2440 # tftp 32000000 uImage;nand erase.part kernel;

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: 00:0c:29:2a:5c:a5

operating at 100M full duplex mode

Using dm9000 device

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'uImage'.

Load address: 0x32000000

Loading: T ###

###

############################

361.3 KiB/s

done

Bytes transferred = 2317984 (235ea0 hex)

NAND erase.part: mtdparts variable not set, see 'help mtdparts'

incorrect device type in kernel

TQ2440 # mtdpart default

TQ2440 # saveenv

Saving Environment to NAND...

Erasing NAND...

Erasing at 0x40000 -- 100% complete.

Writing to NAND... OK

TQ2440 # nand erase.part kernel

NAND erase.part: device 0 offset 0x200000, size 0x300000

Erasing at 0x4e0000 -- 100% complete.

OK

TQ2440 # nand write 32000000 kernel　　烧写内核

NAND write: device 0 offset 0x200000, size 0x300000

3145728 bytes written: OK

TQ2440 # tftp 30000000 root.bin　确保 yaffs 文件系统是好的

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: 00:0c:29:2a:5c:a5

operating at 100M full duplex mode

Using dm9000 device

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'root.bin'.

Load address: 0x30000000

Loading: T ##

##

##

##

##

##

##########

778.3 KiB/s

done

Bytes transferred = 6825984 (682800 hex)

TQ2440 # nand erase.part rootfs

TQ2440 # nand write.yaffs 30000000 rootfs $filesize

设置 u-boot 启动参数

TQ2440 # set bootargs root=/dev/mtdblock2 rootfstype=yaffs2 init=/linuxrc console=ttySAC0,115200

TQ2440 # set bootcmd nand read 32000000 kernel\;bootm 32000000

TQ2440 # save

TQ2440 # reset

resetting ...

U-Boot 2014.04 (Jun 29 2014 - 08:14:15)

CPUID: 32440001

FCLK: 400 MHz

HCLK: 100 MHz

PCLK: 50 MHz

DRAM: 64 MiB

WARNING: Caches not enabled

Flash: *** failed ***

NAND: 256 MiB

In: serial

Out: serial

Err: serial

Net: dm9000

Hit any key to stop autoboot: 0

NAND read: device 0 offset 0x200000, size 0x300000

3145728 bytes read: OK

Booting kernel from Legacy Image at 32000000 ...

Image Name: Linux-3.8.7

Image Type: ARM Linux Kernel Image (uncompressed)

Data Size: 2317920 Bytes = 2.2 MiB

Load Address: 30008000

Entry Point: 30008000

Verifying Checksum ... OK

Loading Kernel Image ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.

Error: unrecognized/unsupported machine ID (r1 = 0x000000c1).

Available machine support:

ID (hex)　　NAME

000000a8　　SMDK2440

Please check your kernel config and/or bootloader.

不支持的机器 ID

由于我用的是 tq2440 自带的内核，tq2440 的机器 ID 是 168，修改 arch/arm/mach-types.h，添加 TQ2440 的机器 ID

```
00055: #define MACH_TYPE_H7201                  161
00056: #define MACH_TYPE_H7202                  162
00057: #define MACH_TYPE_TQ2440                 168
00058: #define MACH_TYPE_IQ80321                169
```

两种解决办法：

1、 设置环境变量 machid

TQ2440 # set machid a8

TQ2440 # save

2、 修改 board/tq2440/tq2440.c: board_init 函数

gd->bd->bi_arch_number = MACH_TYPE_TQ2440;

解决后重启开发板

resetting ...

U-Boot 2014.04 (Jun 29 2014 - 08:14:15)

CPUID: 32440001

FCLK:　　400 MHz

```
HCLK:           100 MHz
PCLK:            50 MHz
DRAM:    64 MiB
WARNING: Caches not enabled
Flash: *** failed ***
NAND:    256 MiB
In:       serial
Out:      serial
Err:      serial
Net:      dm9000
Hit any key to stop autoboot:    0

NAND read: device 0 offset 0x200000, size 0x300000
  3145728 bytes read: OK
## Booting kernel from Legacy Image at 32000000 ...
      Image Name:      Linux-2.6.30.4-EmbedSky
      Image Type:      ARM Linux Kernel Image (uncompressed)
      Data Size:       2415756 Bytes = 2.3 MiB
      Load Address: 30008000
      Entry Point:    30008000
      Verifying Checksum ... OK
      Loading Kernel Image ... OK
Using machid 0xa8 from environment

Starting kernel ...
```

Uncompressing

Linux.....

done, booting the kernel.

Linux version 2.6.30.4-EmbedSky (pengdl@debian) (gcc version 4.3.3 (Sourcery G++ Lite 2009q1-176)) #1

Sat Jun 21 02:35:07 EDT 2014

CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177

CPU: VIVT data cache, VIVT instruction cache

.....

NET: Registered protocol family 17

RPC: Registered udp transport module.

RPC: Registered tcp transport module.

lib80211: common routines for IEEE802.11 drivers

s3c2410-rtc s3c2410-rtc: setting system clock to 2013-03-03 18:07:17 UTC (1362334037)

yaffs: dev is 32505858 name is "mtdblock2"

yaffs: passed flags ""

yaffs: Attempting MTD mount on 31.2, "mtdblock2"

block 325 is bad

block 330 is bad

block 519 is bad

block 1252 is bad

block 1753 is bad

block 1754 is bad

yaffs_read_super: isCheckpointed 0

VFS: Mounted root (yaffs2 filesystem) on device 31:2.

Freeing init memory: 240K

Warning: unable to open an initial console.

Failed to execute /linuxrc. Attempting defaults...

Kernel panic - not syncing: No init found. Try passing init= option to kernel.

Backtrace:

[<c0048fd4>] (dump_backtrace+0x0/0x10c) from [<c036f630>] (dump_stack+0x18/0x1c)

 r7:00000000 r6:c04e8d40 r5:c04e8700 r4:c04b0248

[<c036f618>] (dump_stack+0x0/0x1c) from [<c036f680>] (panic+0x4c/0x124)

[<c036f634>] (panic+0x0/0x124) from [<c00444f0>] (init_post+0xec/0x178)

 r3:00000000 r2:00000000 r1:c051f000 r0:c043c77c

[<c0044404>] (init_post+0x0/0x178) from [<c000847c>] (kernel_init+0xcc/0xf4)

 r5:c001f860 r4:c001fcc8

[<c00083b0>] (kernel_init+0x0/0xf4) from [<c0059f30>] (do_exit+0x0/0x620)

 r7:00000000 r6:00000000 r5:00000000 r4:00000000

已经挂载上 yaffs2 文件系统，因为 yaffs 文件系统是好的，所以是 u-boot 烧写 yaffs 文件系统的问题

YAFFS 中，文件是以固定大小的数据块进行存储的，块的大小可以是 512 字节、1 024 字节或者 2 048 字节。这种实现依赖于它能够将一个数据块头和每个数据块关联起来。每个文件（包括目录）都有一个数据块头与之相对应，数据块头中保存了 ECC(Error Correction Code)和文件系统的组织信息，用于错误检测和坏块处理。

TQ2440 开发板用的 NAND FLASH 每页 2k

用 UltraEdit 打开刚才烧写的 root.bin 文件和烧到 NAND FLASH 里的数据对比

在 u-boot 中执行

TQ2440 # nand dump 500000

Page 00500000 dump:

        03 00 00 00 01 00 00 00   ff ff 00 00 00 00 00 00

```
                ……
            ff ff ff ff ff ff ff ff    ff ff ff ff ff ff ff ff
OOB:
            ff ff ff ff ff ff ff ff
            ff ff ff ff ff ff ff ff
            ff ff ff ff ff ff ff ff
            ff ff ff ff ff ff ff ff
            ff ff ff ff ff ff ff ff
            aa 9a a7 ff f3 03 ff ff
            ff ff ff ff ff ff ff ff
            ff ff ff ff ff ff ff ff
TQ2440 #
```

通过对比发现烧到 NAND FLASH 里的数据和原文件不一致

查看代码 common/cmd_nand.c

```
#ifdef CONFIG_CMD_NAND_YAFFS
    } else if (!strcmp(s, ".yaffs")) {
        if (read) {
            printf("Unknown nand command suffix '%s'.\n", s);
            return 1;
        }
        ret = nand_write_skip_bad(nand, off, &rwsize,
                    (u_char *)addr,
                    WITH_YAFFS_OOB);
#endif
```

调用函数 drivers/mtd/nand/nand_util.c: nand_write_skip_bad, 注意这里传入了一个标志

WITH_YAFFS_OOB

drivers/mtd/nand/nand_util.c：nand_write_skip_bad()函数

if (!need_skip && !(flags & WITH_DROP_FFS)) {

rval = nand_write (nand, offset, length, buffer);

这里如果没有坏块而且没有指定 WITH_DROP_FFS 标志就执行 nand_write (nand, offset, length, buffer);

而我们需要执行 write_oob(nand, offset, &ops);所以应该加上之前传入的参数

if (!need_skip && !(flags & WITH_DROP_FFS) && !(flags & WITH_YAFFS_OOB))

ops.mode = MTD_OOB_RAW; /* 原来为 AUTO，应该为原始 */

重新编译，烧写新的 u-boot.bin，重新烧写 yaffs 文件系统

TQ2440 # nand erase.part u-boot;tftp 32000000 u-boot.bin;nand write 32000000 u-boot

TQ2440 # reset

TQ2440 # tftp 0x32000000 root.bin;nand erase.part rootfs;nand write.yaffs 0x32000000 rootfs $filesize

dm9000 i/o: 0x20000000, id: 0x90000a46

DM9000: running in 16 bit mode

MAC: 00:0c:29:2a:5c:a5

operating at 100M full duplex mode

Using dm9000 device

TFTP from server 192.168.1.8; our IP address is 192.168.1.6

Filename 'root.bin'.

Load address: 0x32000000

Loading: T ###

###

###

###

###

```
	#################################################################
	#################################################################
	##########
	777.3 KiB/s
done
Bytes transferred = 6825984 (682800 hex)

NAND erase.part: device 0 offset 0x500000, size 0xfb00000
Skipping bad block at  0x02d80000
Skipping bad block at  0x02e20000
Skipping bad block at  0x045c0000
Skipping bad block at  0x0a160000
Skipping bad block at  0x0e000000
Skipping bad block at  0x0e020000
Erasing at 0xffe0000 -- 100% complete.
OK

NAND write: device 0 offset 0x500000, size 0x682800
  6825984 bytes written: OK
TQ2440 # reset
……
s3c2410-rtc s3c2410-rtc: setting system clock to 2013-03-03 18:11:59 UTC (1362334319)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
```

block 325 is bad

block 330 is bad

block 519 is bad

block 1252 is bad

block 1753 is bad

block 1754 is bad

yaffs_read_super: isCheckpointed 0

VFS: Mounted root (yaffs2 filesystem) on device 31:2.

Freeing init memory: 240K

mmc0: new SDHC card at address 1234

mmcblk0: mmc0:1234 SA04G 3.63 GiB

mmcblk0: p1

FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!

Please press Enter to activate this console.

[root@TQ2440 /]#

[root@TQ2440 /]#

十、 添加 NAND FLASH 硬件 ECC

在 tq2440.h 中定义如下宏

#define CONFIG_S3C2440_NAND_HWECC

#ifdef CONFIG_S3C2440_NAND_HWECC

#define CONFIG_SYS_NAND_ECCSIZE 2048

#define CONFIG_SYS_NAND_ECCBYTES 4

#endif

修改 arch/arm/include/asm/arch-s3c24x0/s3c24x0.h：s3c2440_nand 结构体，添加寄存器

```
 u32      nfstat0;
 u32      nfstat1;
 u32 nfmecc0;
 u32 nfmecc1;
 u32 nfsecc;
 u32 nfsblk;
 u32 nfeblk;
```

修改 drivers/mtd/nand/s3c2440_nand.c

```
void s3c2440_nand_enable_hwecc(struct mtd_info *mtd, int mode)
{
 struct s3c2440_nand *nand = s3c2440_get_base_nand();
 debug("s3c2440_nand_enable_hwecc(%p, %d)\n", mtd, mode);
 /* 初始化 ECC */
 writel(readl(&nand->nfcont) | (1 << 4), &nand->nfcont);
}
static int s3c2440_nand_calculate_ecc(struct mtd_info *mtd, const u_char *dat,
                      u_char *ecc_code)
{
 struct s3c2440_nand *nand = s3c2440_get_base_nand();
 writel(readl(&nand->nfcont)| (1 << 5),&nand->nfcont); /* 锁定 main 区 ECC */
 /* 读取寄存器 NFMECC0，该寄存器存放着由硬件生成的 main 区 ECC */
 u32    mecc0;
 mecc0 = readl(&nand->nfmecc0);
 ecc_code[0] = mecc0 & 0xff;
 ecc_code[1] = (mecc0 >> 8) &0xff;
```

```
 ecc_code[2] = (mecc0 >> 16) &0xff;
 ecc_code[3] =(mecc0 >> 24) & 0xff;
 debug("s3c2440_nand_calculate_hwecc(%p,): 0x%02x 0x%02x 0x%02x 0x%02x\n",
            mtd , ecc_code[0], ecc_code[1], ecc_code[2], ecc_code[3]);
 return 0;
}
static int s3c2440_nand_correct_data(struct mtd_info *mtd, u_char *dat,
                        u_char *read_ecc, u_char *calc_ecc)
{
 struct    s3c2440_nand *nand = s3c2440_get_base_nand();
 u32    meccdata0, meccdata1, estat0, err_byte_addr;
 int    ret = -1;
 u8    repaired;
 meccdata0 = (read_ecc[1] << 16) | read_ecc[0];
 meccdata1 = (read_ecc[3] << 16) | read_ecc[2];
 writel(meccdata0, &nand->nfeccd0);
 writel(meccdata1, &nand->nfeccd1);
 /*Read ecc status */
 estat0 = readl(&nand->nfstat0);
 switch(estat0 & 0x3) {
 case    0: /* No error */
      ret= 0;
      break;
 case    1:
      /*
```

```
        * 1 bit error (Correctable)
        * (nfestat0 >> 7) & 0x7ff       :error byte number
        * (nfestat0 >> 4) & 0x7          :error bit number
        */
        err_byte_addr = (estat0 >> 7) & 0x7ff;
        repaired = dat[err_byte_addr] ^ (1 << ((estat0 >> 4) & 0x7));

        printf("s3c2440_nand_correct_data: 1 bit error detected at byte %ld. Correcting from 0x%02x to 0x%02x...OK\n",
                err_byte_addr, dat[err_byte_addr], repaired);
        dat[err_byte_addr] = repaired;
        ret= 0;
        break;
    case    2: /* Multiple error */
    case    3: /* ECC area error */
        printf("s3c2440_nand_correct_data: ECC uncorrectable errordetected. " "Not correctable.\n");
        ret= -1;
        break;
    }
    return ret;
}
```

在 tq2440.h 中定义 CONFIG_MTD_NAND_VERIFY_WRITE 实现把所写的数据再读取一遍，然后与被写入的数据之间进行比较来判断所写数据的正确性。

```
#define CONFIG_MTD_NAND_VERIFY_WRITE
```

drivers/mtd/nand/nand_base.c：nand_write_page 函数

```
#ifdef CONFIG_MTD_NAND_VERIFY_WRITE
 /* Send command to read back the data */
 chip->cmdfunc(mtd, NAND_CMD_READ0, 0, page);

 if (chip->verify_buf(mtd, buf, mtd->writesize))
     return -EIO;
#endif
```

十一、 最后

最后可以将 malloc 区域的大小重新改成 4M+160KB，修改 tq2440.h：

#define CONFIG_SYS_MALLOC_LEN (2 * 1024 * 1024)

改为：

#define CONFIG_SYS_MALLOC_LEN (4 * 1024 * 1024)

重新编译，下载运行，正常。

十二、 制作补丁

进入将刚才移植好的 u-boot-2014.04 顶层目录，执行如下命令

pengdl@debian:~/work/tq2440$ mv u-boot-2014.04 u-boot-2014.04_tq2440_god_v2

pengdl@debian:~/work/tq2440$ cd u-boot-2014.04_tq2440_god_v2

pengdl@debian:~/work/tq2440/u-boot-2014.04_tq2440_god_v2$ make distclean

解压未修改过的

pengdl@debian:~/work/tq2440$ tar -xjf u-boot-2014.04.tar.bz2

制作补丁

pengdl@debian:~/work/tq2440$ diff -urwNB u-boot-2014.04 u-boot-2014.04_tq2440_god_v2/ > u-boot-2014.04_tq2440_v2.patch

打补丁

进入未修改的 u-boot-2014.04 源码目录

```
pengdl@debian:~/work/tq2440/u-boot-2014.04$ patch -p1 < ../u-boot-2014.04_tq2440_v2.patch
pengdl@debian:~/work/tq2440/u-boot-2014.04$ make tq2440_config
pengdl@debian:~/work/tq2440/u-boot-2014.04$ make
```

十三、 用到的一些命令：

向 NorFlash 中烧写 uboot:

```
protect off all
tftp 0x30000000 u-boot.bin
erase 0 +40000
cp.b 0x30000000 0x0 0x40000
```

向 NandFlash 中烧写 uboot

```
tftp 0x30000000 u-boot.bin
nand erase.part u-boot
#前提是在 tq2440 中做了配置：
#define MTDIDS_DEFAULT  "nand0=tq2440-0"
#define MTDPARTS_DEFAULT   "mtdparts=tq2440-0:1m(u-boot),"     \
                           "1m(params),"      \
                           "3m(kernel)," \
                           "-(rootfs)"
#参见第七步
nand write 0x30000000 u-boot 0x40000
```

向 NandFlash 中烧写内核：

```
tftp 0x30000000 uImage
nand erase.part kernel
```

nand write 0x30000000 kernel 0x300000

向 NandFlash 中烧写 yaffs 格式的 rootfs

tftp 0x30000000 root.bin

nand erase.part rootfs

nand write.yaffs 0x30000000 rootfs $filesize