Uboot-1.1.6移植到mini2440

ver 1.0

History

|  |  |  |  |
| --- | --- | --- | --- |
| Ver | Author | Date | Context |
| 1.0 | Flinn | Oct-17-2018 | Initial |
|  |  |  |  |

# uboot 编译

Toolchaint ： arm-linux-gcc 3.4.5

PC： ubuntu 14.04 64bit

## 解压

tar xvf u-boot-1.1.6.tar.bz2

## 创建工作目录

cd board

cp -r smdk2410/ mini2440

cd mini2440/

mv mini2410.c mini2440.c

vim Makefile

COBJS := mini2440.o flash.o

修改顶层Makefile

mini2440\_config : unconfig

@$(MKCONFIG) $(@:\_config=) arm arm920t mini2440 NULL s3c24x0

新建config/mini2440.h

cd include/configs/

cp smdk2410.h mini2440.h

## 编译

执行 make mini2440\_config

make all

# 移植

## 修改SDRAM

cd board/mini2440

vim lowlevel\_init.S

#define REFCNT 0x4f4

## 修改时钟

目的：  
FCLK=400MHz  
FCLK:HCLK:PCLK =1：4：8  
UPLL=48MHz,即UCLK=48MHz

File ：/board/mini2440/mini2440.c board\_init

#include <common.h>

#include <s3c2410.h>

DECLARE\_GLOBAL\_DATA\_PTR;

#define M\_MDIV 0x5c

#define M\_PDIV 0x1

#define M\_SDIV 0x1

#define U\_M\_MDIV 0x38

#define U\_M\_PDIV 0x2

#define U\_M\_SDIV 0x2

#define S3C2440\_CLKDIV 0x05

static inline void delay (unsigned long loops)

{

\_\_asm\_\_ volatile ("1:\n"

"subs %0, %1, #1\n"

"bne 1b":"=r" (loops):"0" (loops));

}

int board\_init (void)

{

S3C24X0\_CLOCK\_POWER \* const clk\_power = S3C24X0\_GetBase\_CLOCK\_POWER();

S3C24X0\_GPIO \* const gpio = S3C24X0\_GetBase\_GPIO();

clk\_power->CLKDIVN = S3C2440\_CLKDIV;

\_\_asm\_\_(

"mrc p15, 0, r1, c1, c0, 0\n"

"orr r1, r1, #0xc0000000\n"

"mcr p15, 0, r1, c1, c0,0\n"

:::"r1"

);

/\* to reduce PLL lock time, adjust the LOCKTIME register \*/

clk\_power->LOCKTIME = 0xFFFFFF;

/\* configure MPLL \*/

clk\_power->MPLLCON = ((M\_MDIV << 12) + (M\_PDIV << 4) + M\_SDIV);

/\* some delay between MPLL and UPLL \*/

delay (4000);

/\* configure UPLL \*/

clk\_power->UPLLCON = ((U\_M\_MDIV << 12) + (U\_M\_PDIV << 4) + U\_M\_SDIV);

/\* some delay between MPLL and UPLL \*/

delay (8000);

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

gpio->GPACON = 0x007FFFFF;

gpio->GPBCON = 0x00044555;

gpio->GPBUP = 0x000007FF;

gpio->GPCCON = 0xAAAAAAAA;

gpio->GPCUP = 0x0000FFFF;

gpio->GPDCON = 0xAAAAAAAA;

gpio->GPDUP = 0x0000FFFF;

gpio->GPECON = 0xAAAAAAAA;

gpio->GPEUP = 0x0000FFFF;

gpio->GPFCON = 0x000055AA;

gpio->GPFUP = 0x000000FF;

gpio->GPGCON = 0xFF95FFBA;

gpio->GPGUP = 0x0000FFFF;

gpio->GPHCON = 0x002AFAAA;

gpio->GPHUP = 0x000007FF;

/\* arch number of SMDK2410-Board \*/

//gd->bd->bi\_arch\_number = MACH\_TYPE\_SMDK2410;

gd->bd->bi\_arch\_number = MACH\_TYPE\_S3C2440

/\* adress of boot parameters \*/

gd->bd->bi\_boot\_params = 0x30000100;

icache\_enable();

dcache\_enable();

return 0;

}

编译 make all

## 再修改get\_PCLK

Start\_armboot 会调用serial\_init->get\_PCLK

File : cpu/arm920t/s3c24x0/speed.c

static ulong get\_PLLCLK(int pllreg)

{

… …

//return((CONFIG\_SYS\_CLK\_FREQ \* m) / (p << s));

return((CONFIG\_SYS\_CLK\_FREQ \* m \*2) / (p << s));

}

ulong get\_HCLK(void)

{

unsigned long camdiv;

int hdiv = 1;

… …

camdiv = clk\_power->CAMDIVN;

hdiv = (camdiv & (1 << 9)) ? 8 : 4;

//return((clk\_power->CLKDIVN & 0x2) ? get\_FCLK()/2 : get\_FCLK());

return get\_FCLK() / hdiv;

}

注意smdk2410 S3C24X0\_CLOCK\_POWER没有CAMDIVN成员

在include/s3c24x0.h中添加(最后一个元素)

/\* return PCLK frequency \*/

ulong get\_PCLK(void)

{

… …

unsigned long camdiv;

int hdiv =1;

camdiv = clk\_power->CAMDIVN;

hdiv = (camdiv & (1 << 9)) ? 8 : 4 ;

return get\_FCLK() / hdiv / 2 ;

//return((clk\_power->CLKDIVN & 0x1) ? get\_HCLK()/2 : get\_HCLK());

}

## 支持Nor Flash

Mini2440开发板上配置的是Spansion公司的S29AL016J芯片,

SST39VF1601(AMD29LV160DB), 还是Am29LV160DB ？？？

File ： include/config/mini2440.h

开发板使用的是MX29LV160B

#if 0

//#define CONFIG\_AMD\_LV400 1 /\* uncomment this if you have a LV400 flash \*/

#define CONFIG\_AMD\_LV800 1 /\* uncomment this if you have a LV800 flash \*/

#endif

Nor flash操作函数在board/smdk2440/flash.c， 在Makefile中去掉flash.o,默认flash\_init调用drivers/cfi\_flash.c

在include/configs/smdk2440.h中添加

#define CFG\_FLASH\_BASE PHYS\_FLASH\_1

#define CFG\_MONITOR\_BASE PHYS\_FLASH\_1

/\* nor flash \*/

#define CFG\_FLASH\_CFI\_DRIVER 1

#define CFG\_FLASH\_CFI 1

#ifdef CFG\_FLASH\_CFI\_DRIVER

#define CFG\_MAX\_FLASH\_SECT 512

#endif

在include/flash.h  flash\_info\_t添加成员

uchar manufacturer\_id; /\* manufacturer id \*/

ushort device\_id; /\* device id \*/

ushort device\_id2; /\* extended device id \*/

ushort ext\_addr; /\* extended query table address \*/

ushort cfi\_version; /\* cfi version \*/

ushort cfi\_offset; /\* offset for cfi query \*/

ulong addr\_unlock1; /\* unlock address 1 for AMD flash roms \*/

ulong addr\_unlock2; /\* unlock address 2 for AMD flash roms \*/

const char \*name; /\* human-readable name

## 支持xmode

依照loady命令来仿写loadx

File : common/cmd\_load.c

U\_BOOT\_CMD(

loadx, 3, 0, do\_load\_serial\_bin,

"loadx - load binary file over serial line (xmodem mode)\n",

"[ off ] [ baud ]\n"

" - load binary file over serial line"

" with offset 'off' and baudrate 'baud'\n"

);

在do\_load\_serial\_bin中添加loadx的处理

if (strcmp(argv[0],"loady")==0)

{

printf ("## Ready for binary (xmodem) download "

"to 0x%08lX at %d bps...\n",

offset,

load\_baudrate);

addr = load\_serial\_xmodem (offset);

}

else if (strcmp(argv[0],"loady")==0) {

printf ("## Ready for binary (ymodem) download "

"to 0x%08lX at %d bps...\n",

offset,

load\_baudrate);

… …

}

仿照load\_serial\_ymodem 编写load\_serial\_xmodem

static ulong load\_serial\_ymodem (ulong offset);

static ulong load\_serial\_xmodem (ulong offset);

static ulong load\_serial\_xmodem (ulong offset)

{

int size;

char buf[32];

int err;

int res;

connection\_info\_t info;

char xmodemBuf[1024];

ulong store\_addr = ~0;

ulong addr = 0;

size = 0;

info.mode = xyzModem\_xmodem;

res = xyzModem\_stream\_open (&info, &err);

if (!res){

while ((res =

xyzModem\_stream\_read (xmodemBuf, 1024, &err)) > 0)

{

store\_addr = addr + offset;

size += res;

addr += res;

#ifndef CFG\_NO\_FLASH

if (addr2info (store\_addr)) {

int rc;

rc = flash\_write ((char \*) xmodemBuf,

store\_addr, res);

if (rc != 0) {

flash\_perror (rc);

return (~0);

}

} else

#endif

{

memcpy ((char \*) (store\_addr), xmodemBuf,

res);

}

}

}

else {

printf ("%s\n", xyzModem\_error (err));

}

xyzModem\_stream\_close (&err);

xyzModem\_stream\_terminate (false, &getcxmodem);

flush\_cache (offset, size);

printf ("## Total Size = 0x%08x = %d Bytes\n", size, size);

sprintf (buf, "%X", size);

setenv ("filesize", buf);

return offset;

}

重新烧录支持loadx命令下载文件了。

## 支持网卡CS8900(去掉)

CS8900接在2440 BANK3， 位宽16bit

File ： include/configs/smdk2440.h

#define CONFIG\_COMMANDS \

(CONFIG\_CMD\_DFL | \

CFG\_CMD\_CACHE | \

CFG\_CMD\_PING | \

## 支持DM9000网卡

File ： include/configs/mini2440.h

#if 0

#define CONFIG\_DRIVER\_CS8900 1 /\* we have a CS8900 on-board \*/

#define CS8900\_BASE 0x19000300

#define CS8900\_BUS16 1 /\* the Linux driver does accesses as shorts \*/

#endif

#define CONFIG\_DRIVER\_DM9000

#define CONFIG\_DM9000\_BASE 0x20000000

#define DM9000\_IO CONFIG\_DM9000\_BASE

#define DM9000\_DATA (CONFIG\_DM9000\_BASE+4)

#define CONFIG\_DM9000\_USE\_16BIT 1

#define CONFIG\_DM9000\_NO\_SROM 1

注释掉CS8900,重新烧录，配置开发板IP：192.168.1.123 ， 服务器IP（ubuntu）192.168.133

然后ping 192.168.1.133:

mini2440 :> ping 192.168.1.133

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

MAC: 00:80:00:80:00:80

operating at unknown: 15 mode

host 192.168.1.133 is alive

DM9000执行流程

do\_ping

NetLoop

eth\_halt();

eth\_init();

### tftp下载：

set serverip 192.168.1.104 （windows）

save

kernel：

tftp 30000000 uImage

nand erase 0x00060000 0x200000

nand write.jffs2 30000000 0x60000 0x200000

fs：

tftp 30000000 fs\_mini\_mdev.yaffs2

tftp erase root

nand write.jffs2 30000000 0x260000 889bc0

### nfs下载

安装nfs服务

sudo apt-get install nfs-kernel-server portmap

vim /etc/exports

* /home/flinn/mini2440/bin \*(rw,sync,no\_root\_squash)

sudo /etc/init.d/nfs-kernel-server restart

kernel :

nfs 30000000 192.168.1.133:/home/flinn/mini2440/bin/uImage

nand erase 0x60000 0x200000

nand write.jffs2 30000000 0x60000 0x200000

fs:

tftp 30000000 fs\_qtopta.yaffs2

tftp erase root

nand write.jffs2 30000000 0x260000 real

## 支持mtdparts

和u-boot高版本不同，mtdparts命令没有cmd\_mtdparts这么一个单独的文件来实现。

不过，搜索uboot可以在cmd\_jffs2.c里面看到如下代码：

U\_BOOT\_CMD(

mtdparts, 6, 0, do\_jffs2\_mtdparts,

"mtdparts- define flash/nand partitions\n",

"\n"

" - list partition table\n"

"mtdparts delall\n"

" - delete all partitions\n"

"mtdparts del part-id\n"

" - delete partition (e.g. part-id = nand0,1)\n"

"mtdparts add <mtd-dev> <size>[@<offset>] [<name>] [ro]\n"

...}

可知mtdpart命令是在do\_jffs2\_mtdparts函数里面实现的。

再看do\_jffs2\_mtdparts函数：

int do\_jffs2\_mtdparts(cmd\_tbl\_t \*cmdtp, int flag, int argc, char \*argv[])

{

if (argc == 2) { //这里先检测参数的个数，注意，我们通常使用mtdpart命令时是不带参数，下面会在这里做修改

if (strcmp(argv[1], "default") == 0) {

setenv("mtdids", (char \*)mtdids\_default); //mtdids使用mtdids\_default

setenv("mtdparts", (char \*)mtdparts\_default); //mtdparts使用mtdparts，以上两个默认配置都是在配置文件里面实现的,如smdk2410.h

setenv("partition", NULL);

mtdparts\_init();

return 0;

} else if (strcmp(argv[1], "delall") == 0) {

/\* this may be the first run, initialize lists if needed \*/

mtdparts\_init();

setenv("mtdparts", NULL);

/\* devices\_init() calls current\_save() \*/

return devices\_init();

}

}

...

此外，在cmd\_jffs2.c文件的开头，注意有几个宏定义：

#include <linux/ctype.h>

#if (CONFIG\_COMMANDS & CFG\_CMD\_JFFS2)

#include <cramfs/cramfs\_fs.h>

#if (CONFIG\_COMMANDS & CFG\_CMD\_NAND)

#ifdef CFG\_NAND\_LEGACY

#include <linux/mtd/nand\_legacy.h>

#else /\* !CFG\_NAND\_LEGACY \*/

和

#ifdef CONFIG\_JFFS2\_CMDLINE

/\* default values for mtdids and mtdparts variables \*/

#if defined(MTDIDS\_DEFAULT)

static const char \*const mtdids\_default = MTDIDS\_DEFAULT;

#else

#warning "MTDIDS\_DEFAULT not defined!"

static const char \*const mtdids\_default = NULL;

#endif

这里需要定义三个宏还有默认配置：

#define CONFIG\_JFFS2\_CMDLINE 1

#define CONFIG\_JFFS2\_NAND 1

#define MTDIDS\_DEFAULT "nand0=nandflash0"

#define MTDPARTS\_DEFAULT "mtdparts=nandflash0:256k@0(bootloader)," \

"128k(params)," \

"2m(kernel)," \

"-(root)"

#define CONFIG\_COMMANDS \

CFG\_CMD\_JFFS2 | \

以上，编译通过后，并不能立刻启动内核，注意此时的启动参数：

#define CONFIG\_BOOTCOMMAND "nand read.jffs2 0x30007FC0 kernel; bootm 0x30007FC0"  //因为kernel还没有被识别

这里应该改成：

#define CONFIG\_BOOTCOMMAND "nand read.jffs2 0x30007FC0 0x60000 0x200000; bootm 0x30007FC0"

//这里是内核已经被烧写到Nand里面去了，然后使用nand read命令读到sdram的30007fc0地址处，内核在nand里面的地址是0x60000,大小是2M

这里还是不能使用mtdpart命令，原因是上面提到过mtdpart参数的问题

因为我们不使用参数，那么：

if (argc == 2) {

if (strcmp(argv[1], "default") == 0) {

setenv("mtdids", (char \*)mtdids\_default);

setenv("mtdparts", (char \*)mtdparts\_default);

setenv("partition", NULL);

mtdparts\_init();

return 0;

} else if (strcmp(argv[1], "delall") == 0) {

/\* this may be the first run, initialize lists if needed \*/

mtdparts\_init();

setenv("mtdparts", NULL);

/\* devices\_init() calls current\_save() \*/

return devices\_init();

}

}

/\* make sure we are in sync with env variables \*/

if (mtdparts\_init() != 0)

return 1;

if (argc == 1) {

list\_partitions();

return 0;

}

那么，以下四个函数没有执行：

setenv("mtdids", (char \*)mtdids\_default);  
setenv("mtdparts", (char \*)mtdparts\_default);  
setenv("partition", NULL);

mtdparts\_init();  //其中，主要是这个没有被执行

处理措施：有两种

第一个，把参数解析那个去掉，直接执行mtdparts\_init()函数，代码如下：

int do\_jffs2\_mtdparts(cmd\_tbl\_t \*cmdtp, int flag, int argc, char \*argv[])

{

//by Flinn

setenv("mtdids", (char \*)mtdids\_default);

setenv("mtdparts", (char \*)mtdparts\_default);

setenv("partition", NULL);

mtdparts\_init();

if (argc == 2) {

if (strcmp(argv[1], "default") == 0) {

//setenv("mtdids", (char \*)mtdids\_default);

//setenv("mtdparts", (char \*)mtdparts\_default);

//setenv("partition", NULL);

//mtdparts\_init();

return 0;

} else if (strcmp(argv[1], "delall") == 0) {

/\* this may be the first run, initialize lists if needed \*/

mtdparts\_init();

setenv("mtdparts", NULL);

/\* devices\_init() calls current\_save() \*/

return devices\_init();

}

}

第二种，如韦东山所做，在main.c的main\_loop函数里面添加：

#ifdef CONFIG\_JFFS2\_CMDLINE

extern int mtdparts\_init(void); //先执行init函数

if (!getenv("mtdparts"))

{

run\_command("mtdparts default", 0); //再调用默认配置

}

else

{

mtdparts\_init();

}

#endif

## 支持Nand Flash

添加CFG\_CMD\_NAND

File ： include/configs/smdk2440.h

#define CONFIG\_COMMANDS \

(CONFIG\_CMD\_DFL | \

CFG\_CMD\_CACHE | \

CFG\_CMD\_PING | \

CFG\_CMD\_NAND | \

… …

编译（解决错误）

error: `NAND\_MAX\_CHIPS' undeclared here (not in a function)

nand.c:35: error: `CFG\_MAX\_NAND\_DEVICE' undeclared here (not in a function)

nand.c:38: error: `CFG\_NAND\_BASE' undeclared here (not in a function)

nand.c:35: error: storage size of `nand\_info' isn't known

nand.c:37: error: storage size of `nand\_chip' isn't known

nand.c:38: error: storage size of `base\_address' isn't known

在include/configs/smdk2440.h添加三个宏可以解决上述错误：

#define CFG\_NAND\_BASE 0

#define CFG\_MAX\_NAND\_DEVICE 1

#define NAND\_MAX\_CHIPS 1

还剩最后一个错误：

undefined reference to `board\_nand\_init'

定义S3C2440 Nand数据结构

File : include/s3c24x0.h

typedef struct {

S3C24X0\_REG32 NFCONF;

S3C24X0\_REG32 NFCONT;

S3C24X0\_REG32 NFCMD;

S3C24X0\_REG32 NFADDR;

S3C24X0\_REG32 NFDATA;

S3C24X0\_REG32 NFMECCD0;

S3C24X0\_REG32 NFMECCD1;

S3C24X0\_REG32 NFSECCD;

S3C24X0\_REG32 NFSTAT;

S3C24X0\_REG32 NFESTAT0;

S3C24X0\_REG32 NFESTAT1;

S3C24X0\_REG32 NFMECC0;

S3C24X0\_REG32 NFMECC1;

S3C24X0\_REG32 NFSECC;

S3C24X0\_REG32 NFSBLK;

S3C24X0\_REG32 NFEBLK;

} /\*\_\_attribute\_\_((\_\_packed\_\_))\*/ S3C2440\_NAND;

File : include/s3c2410.h

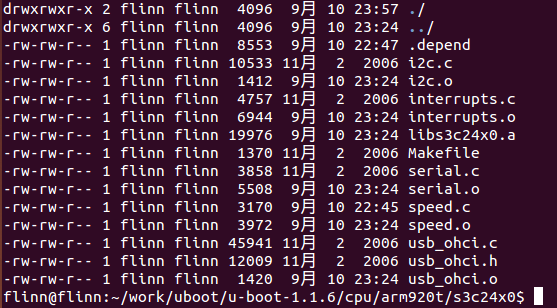
static inline S3C2440\_NAND \* const S3C2440\_GetBase\_NAND(void)

{

return (S3C2440\_NAND \* const)S3C2410\_NAND\_BASE;

}

board\_nand\_init 的实现在cpu/arm920t/s3c24x0/nand\_flash.c



重写nand\_flash.c

/\*

\* s3c2440 nand api

\* copy from linux-2.6.13 drivers/mtd/nand/s3c2410.c

\*/

#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 << 1)

#define S3C2440\_NFSTAT\_READY (1 << 0)

#define S3C2440\_NFCONT\_nFCE (1 << 1)

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:

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)

{

#define TACLS 0

#define TWRPH0 4

#define TWRPH1 2

S3C2440\_NAND \* const s3c2440nand = S3C2440\_GetBase\_NAND();

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

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

}

void board\_nand\_init(struct nand\_chip \*chip)

{

S3C2440\_NAND \* const s3c2440nand = S3C2440\_GetBase\_NAND();

s3c24x0\_nand\_inithw();

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

修改Makefile

COBJS = i2c.o interrupts.o serial.o speed.o \

usb\_ohci.o nand\_flash.o

以上编译可以烧写内核到nand flash。

## 支持烧写yaffs文件系统

增加nand flash可以通过nand write, nand write.jffs2等命令来烧写内核，

tftp 30000000 fs\_mini\_mdev.jffs2

nand erase.part rootfs

nand write.jffs2 30000000 0x00260000 xx

set bootargs console=ttySAC0 root=/dev/mtdblock3 rootfstype=jffs2

烧写正常

烧写yaffs

tftp 30000000 fs\_mini\_mdev.yaffs2 //fs\_qtopia.yaffs2

nand erase.part root

nand write.yaffs 30000000 260000 2f76b40

出现错误：

nand\_write\_ecc: Attempt to write not page aligned data

0 bytes written: ERROR

措施： 增加yaffs2的支持

下面来增加nand write.yaffs

File : common/cmd\_nand.c

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 erase [clean] [off size] - erase `size' bytes from\n"

" offset `off' (entire device if not specified)\n"

"nand bad - show bad blocks\n"

"nand dump[.oob] off - dump page\n"

"nand scrub - really clean NAND erasing bad blocks (UNSAFE)\n"

"nand markbad off - mark bad block at offset (UNSAFE)\n"

"nand biterr off - make a bit error at offset (UNSAFE)\n"

"nand lock [tight] [status] - bring nand to lock state or display locked pages\n"

"nand unlock [offset] [size] - unlock section\n");

添加nand write.yaffs的帮助信息

在do\_nand函数里面增加对write.yaffs的支持。

/\* read write \*/

if (strncmp(cmd, "read", 4) == 0 || strncmp(cmd, "write", 5) == 0)

{

… …

else if ( s != NULL && !strcmp(s, ".yaffs")){

if (read) {

/\* read \*/

nand\_read\_options\_t opts;

memset(&opts, 0, sizeof(opts));

opts.buffer = (u\_char\*) addr;

opts.length = size;

opts.offset = off;

opts.readoob = 1;

opts.quiet = quiet;

ret = nand\_read\_opts(nand, &opts);

} else {

/\* write \*/

nand\_write\_options\_t opts;

memset(&opts, 0, sizeof(opts));

opts.buffer = (u\_char\*) addr;

opts.length = size;

opts.offset = off;

/\* opts.forceyaffs = 1; \*/

opts.noecc = 1;

opts.writeoob = 1;

opts.blockalign = 1;

opts.quiet = quiet;

opts.skipfirstblk = 1;

ret = nand\_write\_opts(nand, &opts);

}

}

… …

}

在nand.h里面加上opts.skipfirstblk以及其实现

struct nand\_write\_options {

u\_char \*buffer; /\* memory block containing image to write \*/

ulong length; /\* number of bytes to write \*/

ulong offset; /\* start address in NAND \*/

int quiet; /\* don't display progress messages \*/

int autoplace; /\* if true use auto oob layout \*/

int forcejffs2; /\* force jffs2 oob layout \*/

int forceyaffs; /\* force yaffs oob layout \*/

int noecc; /\* write without ecc \*/

int writeoob; /\* image contains oob data \*/

int pad; /\* pad to page size \*/

int blockalign; /\* 1|2|4 set multiple of eraseblocks

\* to align to \*/

int skipfirstblk; //这两项添加的

int nocheckbadblk;

};

修改nand\_write\_opts函数，对int skipfirstblk的支持，文件在nand\_util.c的nand\_write\_opts

int skipfirstblk = opts->skipfirstblk;

/\* skip the first good block when wirte yaffs image, by www.100ask.net \*/

if (skipfirstblk) {

mtdoffset += erasesize\_blockalign;

skipfirstblk = 0;

continue;

}

上面由于不用ecc，会出现很多警告信息如：  
Writing data without ECC to NAND-FLASH is not recommended\n  
可以在drivers/nand/nand\_base.c里面的nand\_write\_page函数里面屏蔽掉：  
case NAND\_ECC\_NONE:  
//printk (KERN\_WARNING "Writing data without ECC to NAND-FLASH is not recommended\n");  
this->write\_buf(mtd, this->data\_poi, mtd->oobblock);

nand write.yaffs 30000000 260000 889bc0 烧写yaffs文件系统时：

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

Writing data without ECC to NAND-FLASH is not recommended

## 修改默认配置

File ： include/configs/smdk2440.h

1. Linux启动信息
   1. #define CONFIG\_SETUP\_MEMORY\_TAGS 1 /\* 向内核传递内存分布信息 \*/
   2. #define CONFIG\_CMDLINE\_TAG 1 /\* 向内核传递命令行参数 \*/
   3. /\* 默认命令行参数 \*/
   4. #define CONFIG\_BOOTARGS “noinitrd root=/dev/mtdblock 2 init=/linuxrc console=ttySAC0”
2. 自动启动命令
   1. #define CONFIG\_BOOTDELAY 3
   2. #define CONFIG\_BOOTCOMMAND “nboot 0x32000000 0 0; bootm 0x32000000”
3. 默认网络设置

(略)

## 烧写

device nand0 <nandflash0>, # parts = 4

#: name size offset mask\_flags

0: bootloader 0x00040000 0x00000000 0

1: params 0x00020000 0x00040000 0

2: kernel 0x00200000 0x00060000 0

3: root 0x0fda0000 0x00260000 0

kernel ：

tftp 30000000 uImage

nand erase kernel

// nand erase 0x60000 0x200000

nand write.jffs2 30000000 kernel (0x60000 0x200000)

jffs fs:

tftp 30000000 fs\_mini\_mdev.jffs2

nand erase root

nand write.jffs2 30000000 0x00260000 15787a8

//bootargs=noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0

set bootargs console=ttySAC0 root=/dev/mtdblock3 rootfstype=jffs2

yaffs fs:

tftp 30000000 fs\_mini\_mdev.yaffs2

nand erase root

nand write.yaffs 30000000 260000 889bc0

## 支持nand启动

以上不支持nand启动，原因是start.S里面

relocate: /\* relocate U-Boot to RAM \*/

adr r0, \_start /\* r0 <- current position of code \*/

ldr r1, \_TEXT\_BASE /\* test if we run from flash or RAM \*/

cmp r0, r1 /\* don't reloc during debug \*/

beq stack\_setup

ldr r2, \_armboot\_start

ldr r3, \_bss\_start

sub r2, r3, r2 /\* r2 <- size of armboot \*/

add r2, r0, r2 /\* r2 <- source end address \*/

并没有作nor/nand区分

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT

bl cpu\_init\_crit

#endif

// 把stack setup 放前面初始化栈

/\* Set up the stack \*/

stack\_setup:

ldr r0, \_TEXT\_BASE /\* upper 128 KiB: relocated uboot \*/

sub r0, r0, #CFG\_MALLOC\_LEN /\* malloc area \*/

sub r0, r0, #CFG\_GBL\_DATA\_SIZE /\* bdinfo \*/

#ifdef CONFIG\_USE\_IRQ

sub r0, r0, #(CONFIG\_STACKSIZE\_IRQ+CONFIG\_STACKSIZE\_FIQ)

#endif

sub sp, r0, #12 /\* leave 3 words for abort-stack \*/

然后

ldr r2, \_armboot\_start

ldr r3, \_bss\_start

sub r2, r3, r2 /\* r2 <- size of armboot \*/

copy\_code\_to\_ram();

#if 0

add r2, r0, r2 /\* r2 <- source end address \*/

copy\_loop:

ldmia r0!, {r3-r10} /\* copy from source address [r0] \*/

stmia r1!, {r3-r10} /\* copy to target address [r1] \*/

cmp r0, r2 /\* until source end addreee [r2] \*/

ble copy\_loop

#endif

在board/mini2440/添加文件boot\_init.c

#include <common.h>

#include <s3c2410.h>

#define BUSY 1

#define NAND\_SECTOR\_SIZE 512

#define NAND\_BLOCK\_MASK (NAND\_SECTOR\_SIZE - 1)

#define NAND\_SECTOR\_SIZE\_LP 2048

#define NAND\_BLOCK\_MASK\_LP (NAND\_SECTOR\_SIZE\_LP - 1)

void nand\_init\_ll(void);

void nand\_read\_ll(unsigned char \*buf, unsigned long start\_addr, int size);

static void nand\_reset(void);

static void wait\_idle(void);

static void nand\_select\_chip(void);

static void nand\_deselect\_chip(void);

static void write\_cmd(int cmd);

static void write\_addr(unsigned int addr);

static unsigned char read\_data(void);

static void s3c2440\_nand\_reset(void);

static void s3c2440\_wait\_idle(void);

static void s3c2440\_nand\_select\_chip(void);

static void s3c2440\_nand\_deselect\_chip(void);

static void s3c2440\_write\_cmd(int cmd);

static void s3c2440\_write\_addr(unsigned int addr);

static unsigned char s3c2440\_read\_data(void);

static void s3c2440\_nand\_reset(void)

{

s3c2440\_nand\_select\_chip();

s3c2440\_write\_cmd(0xff); // 澶嶄綅鍛戒护

s3c2440\_wait\_idle();

s3c2440\_nand\_deselect\_chip();

}

static void s3c2440\_wait\_idle(void)

{

int i;

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

volatile unsigned char \*p = (volatile unsigned char \*)&s3c2440nand->NFSTAT;

while(!(\*p & BUSY))

for(i=0; i<10; i++);

}

static void s3c2440\_nand\_select\_chip(void)

{

int i;

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

s3c2440nand->NFCONT &= ~(1<<1);

for(i=0; i<10; i++);

}

static void s3c2440\_nand\_deselect\_chip(void)

{

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

s3c2440nand->NFCONT |= (1<<1);

}

/\* 鍙戝嚭鍛戒护 \*/

static void s3c2440\_write\_cmd(int cmd)

{

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

volatile unsigned char \*p = (volatile unsigned char \*)&s3c2440nand->NFCMD;

\*p = cmd;

}

/\* 鍙戝嚭鍦板潃 \*/

static void s3c2440\_write\_addr(unsigned int addr)

{

int i;

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

volatile unsigned char \*p = (volatile unsigned char \*)&s3c2440nand->NFADDR;

\*p = addr & 0xff;

for(i=0; i<10; i++);

\*p = (addr >> 9) & 0xff;

for(i=0; i<10; i++);

\*p = (addr >> 17) & 0xff;

for(i=0; i<10; i++);

\*p = (addr >> 25) & 0xff;

for(i=0; i<10; i++);

}

/\* 鍙戝嚭鍦板潃 \*/

static void s3c2440\_write\_addr\_lp(unsigned int addr)

{

int i;

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

volatile unsigned char \*p = (volatile unsigned char \*)&s3c2440nand->NFADDR;

int col, page;

col = addr & NAND\_BLOCK\_MASK\_LP;

page = addr / NAND\_SECTOR\_SIZE\_LP;

\*p = col & 0xff; /\* Column Address A0~A7 \*/

for(i=0; i<10; i++);

\*p = (col >> 8) & 0x0f; /\* Column Address A8~A11 \*/

for(i=0; i<10; i++);

\*p = page & 0xff; /\* Row Address A12~A19 \*/

for(i=0; i<10; i++);

\*p = (page >> 8) & 0xff; /\* Row Address A20~A27 \*/

for(i=0; i<10; i++);

\*p = (page >> 16) & 0x03; /\* Row Address A28~A29 \*/

for(i=0; i<10; i++);

}

/\* 璇诲彇鏁版嵁 \*/

static unsigned char s3c2440\_read\_data(void)

{

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

volatile unsigned char \*p = (volatile unsigned char \*)&s3c2440nand->NFDATA;

return \*p;

}

static void nand\_reset(void)

{

s3c2440\_nand\_reset();

}

static void wait\_idle(void)

{

s3c2440\_wait\_idle();

}

static void nand\_select\_chip(void)

{

s3c2440\_nand\_select\_chip();

}

static void nand\_deselect\_chip(void)

{

s3c2440\_nand\_deselect\_chip();

}

static void write\_cmd(int cmd)

{

s3c2440\_write\_cmd(cmd);

}

static void write\_addr(unsigned int addr)

{

s3c2440\_write\_addr(addr);

}

static void write\_addr\_lp(unsigned int addr)

{

s3c2440\_write\_addr\_lp(addr);

}

static unsigned char read\_data(void)

{

return s3c2440\_read\_data();

}

void nand\_read\_ll\_lp(unsigned char \*buf, unsigned long start\_addr, int size)

{

int i, j;

if ((start\_addr & NAND\_BLOCK\_MASK\_LP) || (size & NAND\_BLOCK\_MASK\_LP)) {

return ;

}

nand\_select\_chip();

for(i=start\_addr; i < (start\_addr + size);) {

write\_cmd(0);

/\* Write Address \*/

write\_addr\_lp(i);

write\_cmd(0x30);

wait\_idle();

for(j=0; j < NAND\_SECTOR\_SIZE\_LP; j++, i++) {

\*buf = read\_data();

buf++;

}

}

nand\_deselect\_chip();

return ;

}

static void nand\_init\_ll(void)

{

S3C2440\_NAND \* s3c2440nand = (S3C2440\_NAND \*)0x4e000000;

#define TACLS 0

#define TWRPH0 3

#define TWRPH1 0

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

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

nand\_reset();

}

static int is\_boot\_from\_nor(void)

{

volatile unsigned int \*pdw = (volatile unsigned int \*)0;

unsigned int dwVal;

dwVal = \*pdw;

\*pdw = 0x12345678;

if (\*pdw != 0x12345678)

{

return 1;

}

else

{

\*pdw = dwVal;

return 0;

}

}

int copy\_code\_to\_ram(unsigned long start\_addr, unsigned char \*buf, int size)

{

unsigned int \*pdwDest;

unsigned int \*pdwSrc;

int i;

if(is\_boot\_from\_nor())

{

pdwDest = (unsigned int \*)buf;

pdwSrc = (unsigned int \*)start\_addr;

for (i = 0; i < size / 4; i++)

{

pdwDest[i] = pdwSrc[i];

}

return 0;

}

else

{

// from nand

nand\_init\_ll();

nand\_read\_ll\_lp(buf, start\_addr, (size + NAND\_BLOCK\_MASK\_LP)&~(NAND\_BLOCK\_MASK\_LP));

return 0;

}

}

### 烧写到nand里面

tftp 30000000 u-boot.bin

nand erase bootloader

nand write.jffs2 30000000 bootloader

## 制作补丁

mv linux-2.6.22.6 linux-2.6.22.6\_Flinn

tar xvf linux-2.6.22.6.tar.bz

diff -urNwB linux-2.6.22.6 linux-2.6.22.6\_Flinn > linux-2.6.22.6\_Flinn.diff

## 打补丁

.diff 和linux-kernel在同一目录

cd linux-2.6.22

patch –p1 ../\*.diff