君正®

T41 SFC 开发指南

Date: 2022-08

Trademarks and Permissions

Disclaimer

All the deliverables and data in this folder serve only as a reference for customer development. Please read through this disclaimer carefully before you use the deliverables and data in this folder. You may use the deliverables in this folder or not. However, by using the deliverables and data in this folder, you agree to accept all the content in this disclaimer unconditional and irrevocable. If you do not find the content in this disclaimer reasonable, you shall not use the deliverables and data in this folder.

The deliverables and data in this folder are provided "AS IS" without representations, guarantees or warranties of any kind (either express or implied). To the maximum extent permitted by law, Ingenic Semiconductor Co., Ltd (Ingenic) provides the deliverables and data in this folder without implied representations, guarantees or warranties, including but not limited to implied representations, guarantees and warranties of merchantability, non-infringement, or fitness for a particular purpose. Deviation of the data provided in this folder may exist under different test environments.

Ingenic takes no liability or legal responsibility for any design and development error, incident, negligence, infringement, and loss (including but not limited to any direct, indirect, consequential, or incidental loss) caused by the use of data in this folder. Users shall be responsible for all risks and consequences caused by the use of data in this folder.

北京君正集成电路股份有限公司

地址：北京市海淀区西北旺东路 10 号院东区 14 号楼君正大厦

电话:**(86-10)56345000**

传真:**(86-10)56345001**

Http: //www.ingenic.c

前言

概述

本文档主要是对 T41 中的 Nor/Nand Flash 的添加介绍和配置说明。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
T41	

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修订章节
2022-8	1.0	第一次正式版本发布

目录

1 添加新的 Nor flash

注意：一款新的 norflash 需要在 uboot 和 kernel 中同步添加。

1.1 uboot 中添加 nor flash 配置

步骤 1：确认 ID

相同厂商制作的 flash 会有相同的厂商 ID，不同厂商制作的 flash 通常厂商 ID 会不同，添加 flash 前先确定是否已经添加了对应的厂商 ID。

以 norflash IS25LP128F 为例（默认读取 ID 的命令是 9F），其厂商 ID 为 9Dh，设备 ID 为 6018h。

图 1-1 厂商 ID 信息

Manufacturer ID		**(MF7-MF0)**	
ISSI Serial Flash		9Dh	
Instruction	**ABh**	**90h**	**9Fh**
Part Number	Device ID (ID7-ID0)		Memory Type + Capacity (ID15-ID0)
IS25LP128F	17h		6018h
IS25WP128F	17h		7018h

```
$ vi drivers/mtd/spi/spi_flash.c
```

如下图所示，copy 一份现有的配置，将第二个参数改成厂商 ID，如 IS25LP128F 的厂商 ID 为 9d，故添加 0x9d 的配置；如果厂商 ID 已经存在，则跳过此步骤。

```
458 static const struct {
459     const u8 shift;
460     const u8 idcode;
461     struct spi_flash *(*probe) (struct spi_slave *spi, u8 *idcode);
462 } flashes[] = {
463     /* Keep it sorted by define name */
464 #ifdef CONFIG_SPI_FLASH_INGENIC
465 #ifdef CONFIG_SPI_FLASH_INGENIC_NAND
466     { 0, 0xc8, spi_flash_probe_ingenic_nand, },
467 #endif
468     { 0, 0x52, spi_flash_probe_ingenic, },
469     { 0, 0x5e, spi_flash_probe_ingenic, },
470     { 0, 0xe0, spi_flash_probe_ingenic, },
471     { 0, 0xc2, spi_flash_probe_ingenic, },
472     { 0, 0xef, spi_flash_probe_ingenic, },
473     { 0, 0x1c, spi_flash_probe_ingenic, },
474     { 0, 0xF8, spi_flash_probe_ingenic, },
475     { 0, 0x20, spi_flash_probe_ingenic, },
476     { 0, 0x68, spi_flash_probe_ingenic, },
477     { 0, 0xa1, spi_flash_probe_ingenic, },
478     { 0, 0x0b, spi_flash_probe_ingenic, },
479     { 0, 0x85, spi_flash_probe_ingenic, },
480     { 0, 0xba, spi_flash_probe_ingenic, },
481     { 0, 0x9d, spi_flash_probe_ingenic, },
482
~/work/isvp/work/opensource/uboot/drivers/mtd/spi/spi_flash.c [HEX=0][481,43,6
```

图 1-2 添加厂商信息

步骤 2：添加 flash 属性参数

```
$ vi drivers/spi/jz_spi.h
```

找到 jz_spi_support_table 数组，按照已有的 flash 格式复制粘贴一份，主要修改 name, id_manufactory, sector_size, size。

以添加一个 16M 的 flash IS25LP128F 为例：

```
{
    .name          = "IS25LP128F",
    .id_manufactory = 0x9d6018,
    .page_size      = 256,
    .sector_size     = (64 * 1024),
    .addr_size = 3,
    .size          = (16 * 1024 * 1024),
    .quad_mode = {
        .dummy_byte = 8,
```

```
            .RDSR_CMD = CMD_RDSR,
            .WRSR_CMD = CMD_WRSR,
            .RDSR_DATE = 0x40,
            .RD_DATE_SIZE = 1,
            .WRSR_DATE = 0x40,
            .WD_DATE_SIZE = 1,
            .cmd_read = CMD_QUAD_READ,
#ifdef CONFIG_JZ_SFC
            .sfc_mode = TRAN_CONF1_SPI_QUAD,
#endif
        },
    },
```

参数介绍：

1) name：spi nor flash 的名字；
2) id_manufactory：spi flash 的 id，厂家 ID+设备 ID，对于 IS25LP128F 即为 0x9d6018；
3) page_size：spi flash 写操作的页大小；
4) sector_size：flash 擦的大小（建议设置 64k，如果不支持请设成 32k）；
5) addr_size：访问 flash 的地址长度(byte)，16MB 及以下为 3，大于 16MB 通常为 4；
6) size：spi nor norflash 的总大小(byte)；
7) quad_mode：四线模式，需要根据 spec 定义设置里面的参数；
8) dummy_byte：dummy clocks 的个数（根据 spec 配置）；
9) RDSR_CMD/WRSR_CMD：读写包含 QE 位状态寄存器的命令，如下图：

RDSR_CMD 是 0x05,WRSR_CMD 是 0x01；

RDSR	Read Status Register	SPI QPI	05h	Data out
WRSR	Write Status Register	SPI QPI	01h	Data in

10) RDSR_DATE/WDSR_DATE：是用来读/写 QE 位的值，如图 QE 是第六位，故为 0x40；

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	SRWD	QE	BP3	BP2	BP1	BP0	WEL	WIP
Default	0	0	0	0	0	0	0	0

11) RD_DATE_SIZE/WD_DATE_SIZE：发送命令的长度大小（字节）；

12) cmd_read：四线模式快速读取数据的指令，以 IS25LP128F 为例，该命令为 0x6b；

FRQO	Fast Read Quad Output (3-byte Address)	SPI	6Bh	A <23:16>	A <15:8>	A <7:0>	Dummy(1) Byte	Quad Data out	

13) sfc_mode：对应 4 线模式控制器的传输值，默认为 TRAN_CONF1_SPI_QUAD。

（注：以上所有参数可以在 spec 中找到）

1.2 kernel 中添加 nor flash 配置

```
$ vi drivers/mtd/devices/ingenic_sfc_v2/spinor.h
```

找到 spi_nor_info_table，在里面 copy 一份现有的配置，主要修改 chip_size,chip_erase_cmd，如果需要设置四线模式，注意 quad_set 和 quad_get，以 IS25LP128F 为例：

```
    {
        .name = "IS25LP128F",
        .id = 0x9d6018,
#ifdef CONFIG_SPI_STANDARD_MODE
        .quad_ops_mode = 0,
#else
        .quad_ops_mode = 1,
#endif
        /* addr_ops_mode: en4byte or addr_len > 3, set 1 */
        .addr_ops_mode = 0,
        .tCHSH = 5,
        .tSLCH = 5,
        .tSHSL_RD = 20,
        .tSHSL_WR = 20,
        .chip_size = 16 * 1024 * 1024,
        .page_size = 256,
        .erase_size = 32 * 1024,
        .addr_len = 3,
        .chip_erase_cmd = SPINOR_OP_CHIP_ERASE,
        .quad_set = {
            .cmd = SPINOR_OP_WRSR,
            .bit_shift = 6,
        },
        .quad_get = {
            .cmd = SPINOR_OP_RDSR,
            .bit_shift = 6,
        },
    },
```

图 1-3 Flash 信息图

参数介绍：

1) name：flash 名称
2) Id：flash 的 id, 厂家 ID+设备 ID
3) quad_ops_mode：4 线模式的开关
4) addr_ops_mode：地址 4byte 模式，通常 addr_len 为 4 时开启
5) tCHSH/tSLCH/tSHSL_RD/tSHSL_WR 时序配置，可以参考 flash spec
6) chip_size：flash 整体大小
7) page_size：flash 页大小
8) erase_size：擦除大小（建议设置 64k，如果不支持请设成 32k）
9) addr_len：访问 flash 的地址长度(byte)，16MB 及以下为 3，大于 16MB 通常为 4
10) chip_erase_cmd：擦除整个 flash 的命令，根据 spec 配置

11) quad_set/quad_get：

a. cmd 表示读写 QE 寄存器的命令

b. bit_shift 表示 QE 的偏移位

2 添加新的 Nand flash

2.1 spl 中添加 nand flash 配置

步骤 1：确认 ID

添加的目录为 tools/ingenic-tools/nand_device/，首先确认该 NAND 的厂商 ID 是否已经支持，当该 NAND 的厂商 ID 不支持时，

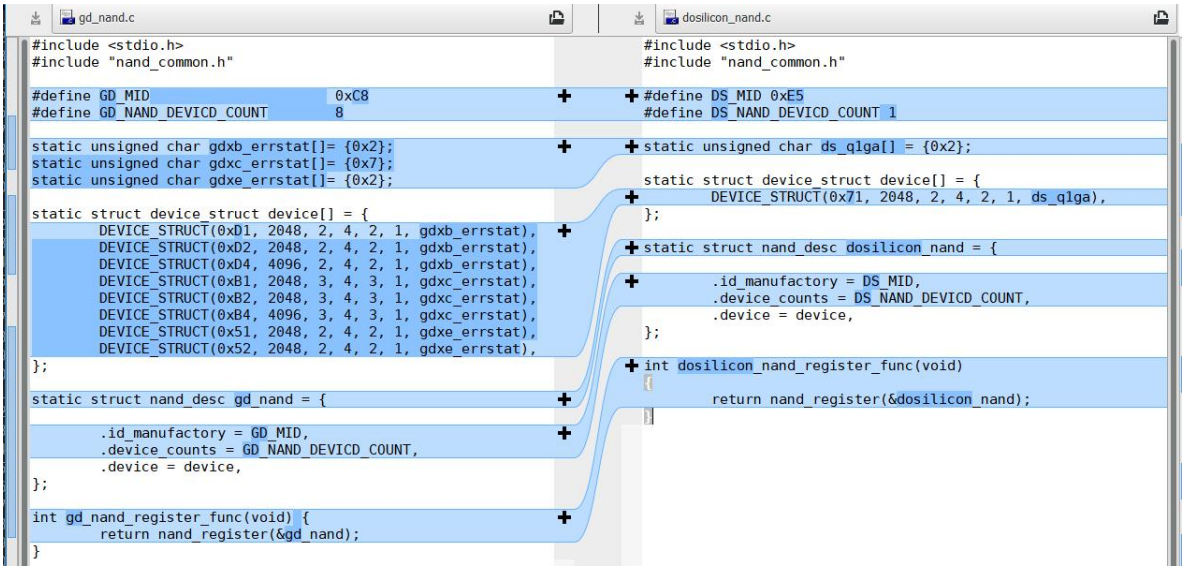
以 dosilicon 东芯 nand 为例：

拷贝一份已有 nand 配置文件(gd_nand.c)，修改其中一些参数内容。

```
$ cp gd_nand.c dosilicon_nand.c

$ vi dosilicon_nand.c
```

```c
#include <stdio.h>
#include "nand_common.h"

#define GD_MID              0xC8
#define GD_NAND_DEVICD_COUNT        8

static unsigned char gdxb_errstat[]= {0x2};
static unsigned char gdxc_errstat[]= {0x7};
static unsigned char gdxe_errstat[]= {0x2};

static struct device_struct device[] = {
    DEVICE_STRUCT(0xD1, 2048, 2, 4, 2, 1, gdxb_errstat),
    DEVICE_STRUCT(0xD2, 2048, 2, 4, 2, 1, gdxb_errstat),
    DEVICE_STRUCT(0xD4, 4096, 2, 4, 2, 1, gdxb_errstat),
    DEVICE_STRUCT(0xB1, 2048, 3, 4, 3, 1, gdxc_errstat),
    DEVICE_STRUCT(0xB2, 2048, 3, 4, 3, 1, gdxc_errstat),
    DEVICE_STRUCT(0xB4, 4096, 3, 4, 3, 1, gdxc_errstat),
    DEVICE_STRUCT(0x51, 2048, 2, 4, 2, 1, gdxe_errstat),
    DEVICE_STRUCT(0x52, 2048, 2, 4, 2, 1, gdxe_errstat),
};

static struct nand_desc gd_nand = {

    .id_manufactory = GD_MID,
    .device_counts = GD_NAND_DEVICD_COUNT,
    .device = device,
};

int gd_nand_register_func(void) {
    return nand_register(&gd_nand);
}
```

需要修改的内容如下



参数介绍：

DENSITY	VCC	1st Manufactory ID	2nd Device ID
1Gbit	3.3V	E5h	71h
	1.8V	E5h	21h

1) DS_MID：厂商 ID

2) DS_NAND_DEVICD_COUNT：表示支持该系列 nand 的个数（相同厂商 ID，但是 deviceID 不同）。

3) ds_q1ga：表示 nand 发生位翻转，且不可恢复时候 ECC 值。

SR[5:4] (ECC_S1, ECC_S0)	ECC Status	The bit shows the status of ECC as below: 00b = 0 bit error 01b = 1 ~ 4 bits error and been corrected. 10b = More than 4-bit error and not corrected. 11b = Reserved The value of ECC_Sx (S1:S0) bits will be clear as "00b" by Reset command or at the start of the Read operation. After a valid Read operation completion, the bit will be updated to reflect the ECC status of the current valid Read operation. The ECC_Sx (S1:S0) value reflects the ECC status of the content of the page 0 of block 0 after a power-on reset. If the internal ECC is disabled by the Set feature command, the ECC_ Sx(S1:S0) are invalid.

4) DEVICE_STRUCT(0x71, 2048, 2, 4, 2, 1, ds_q1ga)：

```
#define DEVICE_STRUCT(id, pagesize, addrlen, bit, bitcounts,  eccstatcount, err) {  \
        .device_id = id,    \
        .page_size = pagesize,  \
        .addr_len = addrlen,    \
        .ecc_bit = bit, \
        .bit_counts = bitcounts,    \
        .eccstat_count = eccstatcount, \
        .eccerrstatus = err,    \
}
```

a. device_id：设备 ID

b. page_size：页大小

c. addr_len：读写数据的地址长度

d. ecc_bit：状态寄存器中 ECC 的最低偏移位

a. bit_counts：状态寄存器中 ECC 所占 bit 数

b. eccstat_count：ECC 错误状态的个数

c. eccerrstatus：储存 ECC error status 内容的地址（通常是无法恢复或者意料之外的）

注：如果厂商 ID 已经支持，只需要在该文件中添加该设备的参数即可。

步骤 2：2G NAND 可能会涉及 plane 选择（如果没有可跳过此步骤）

例如芯天下 XT26G02E

```
column |= (((page >> 6) & 1) << 12);
```

0：plane 表示偶数 block

1：plane 表示奇数 block

```
 if(curr_device->device_id == 0x20 || curr_device->device_id == 0x22)/*MXIC 2G plane select*/
     column |= (((page >> 6) & 1) << 12);
ndef CONFIG_SPI_STANDARD
 SFC_SEND_COMMAND(&sfc, CMD_FR_CACHE_QUAD, len, column, curr_device->addrlen, 8, 1, 0);
se
 SFC_SEND_COMMAND(&sfc, CMD_FR_CACHE, len, column, curr_device->addrlen, 8, 1, 0);
ndif
 sfc_read_data((unsigned int *)dst_addr, len);

 if (!oob_flag && !(page % CONFIG_SPI_NAND_PPB)) {
     oob_flag = 1;
     goto read_oob;
 } else if (oob_flag) {
/tyu/isvp/opensource/uboot/common/spl/spl_sfc_nand.c[POS=193,4][46%]29/05/21 - 11:49
```

Figure1. Array Organization

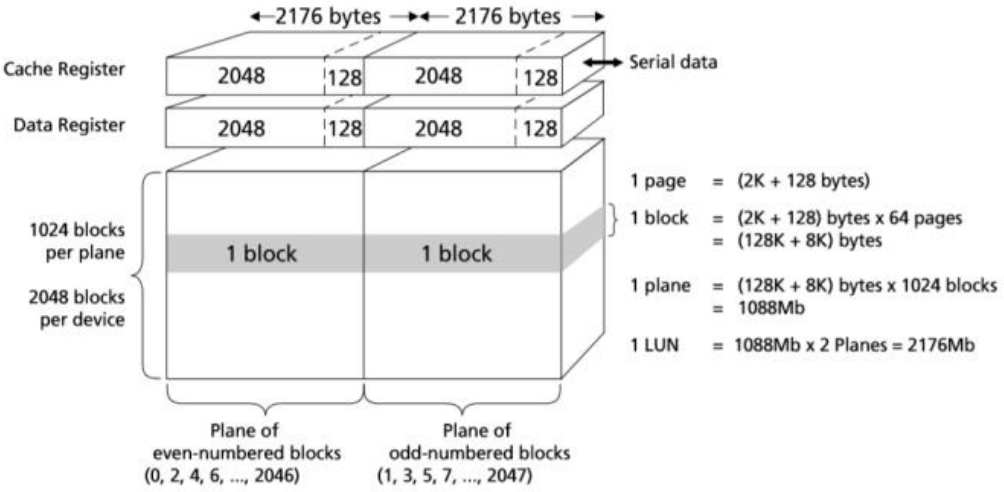
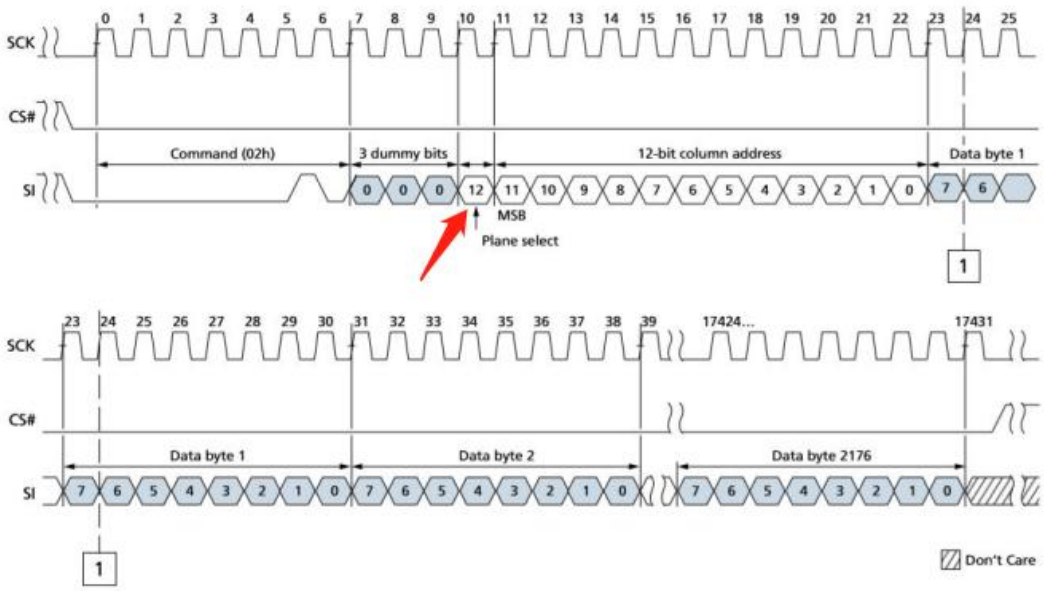


Figure14. PROGRAM LOAD (02h) Timing



Note: 1. Plane select is a dummy bit in 1Gb device.

2.2 uboot 中添加 nand flash 配置

步骤 1：添加配置文件

在 drivers/mtd/devices/jz_sfc_v1/nand_device/这个目录下创建相应 flash 厂商的 c 文件。（例如：foresee_nand.c，添加对江波龙 spinand 的支持）

```
$ cd drivers/mtd/devices/jz_sfc_v1/nand_device/
```

Copy 任意一个已支持厂商文件的内容到新添加 flash 文件中，并将新文件中定义的所有描述符名称进行修改.

```
$ cp xtx_nand.c foresee_nand.c
```

步骤 2：修改参数

1. 对文件标红的地方进行专门修改：（以 foresee_nand.c 为例）

A 处标红： 支持此 flash 厂商的 device 数目，目前只支持江波龙的 FS35ND01G 这一款设备，故该值为 1，如果后面添加了江波龙厂商的其他 nand 设备，这里也要同步修改。

```
#include <malloc.h>
#include <linux/mtd/partitions.h>
#include "../jz_sfc_nand.h"
#include "nand_common.h"

#define FS_DEVICES_NUM      A. 1
#define TSETUP        B. 5
#define THOLD           5
#define TSHSL_R         20
#define TSHSL_W         20

static struct jz_nand_base_param fs_param[FS_DEVICES_NUM] = {

        [0] = {
                /*FS35ND01G*/
            C.  .pagesize = 2 * 1024,
                .blocksize = 2 * 1024 * 64,
                .oobsize = 64,
                .flashsize = 2 * 1024 * 64 * 1024,

                .tSETUP   = TSETUP,
                .tHOLD    = THOLD,
                .tSHSL_R = TSHSL_R,
                .tSHSL_W = TSHSL_W,

                .ecc_max = 0x4,
                .need_quad = 1,
        },

};

static struct device_id struct device_id[FS_DEVICES_NUM] = {
      D. DEVICE_ID_STRUCT(0xA1, "FS35ND01G", &fs_param[0]),
};
```

B 处标红： 当前支持的 flash device 的 CS 时序（添加需要查看 device 数据手册， 如下图位置）。

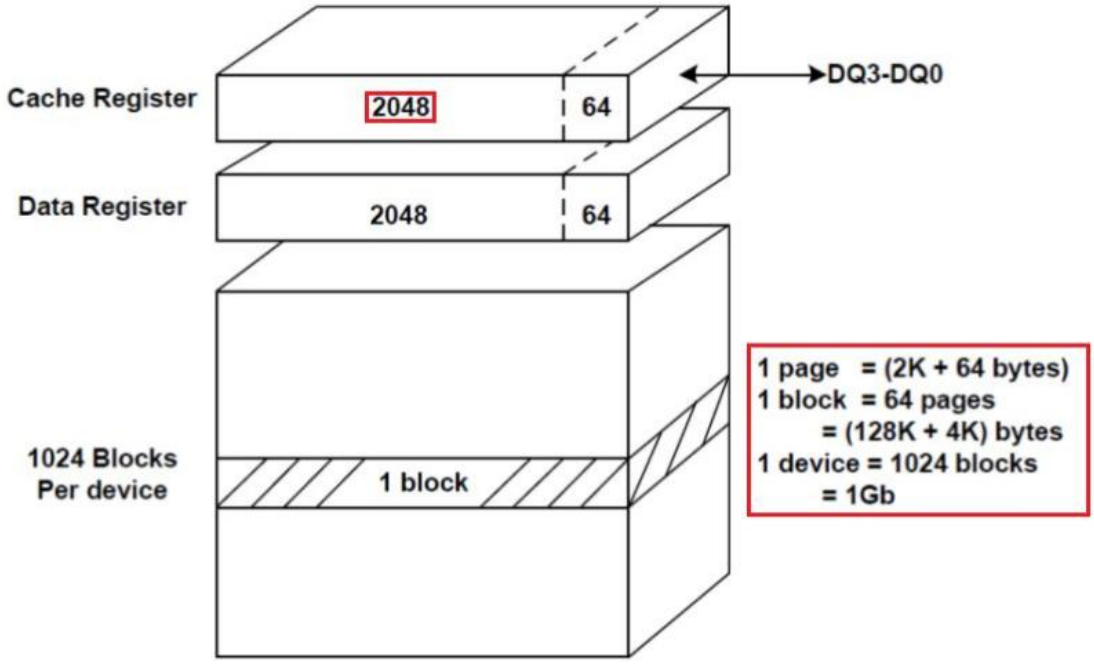
tSLCH	CS# Active Setup Time	5			ns
tCHSH	CS# Active Hold Time	5			ns
tSHCH	CS# Not Active Setup Time	5			ns
tCHSL	CS# Not Active Hold Time	5			ns
tSHSL/tCS	CS# High Time	20			ns

a. TSETUP(tSLCH)：CS# Active Setup Time

b. THOLD (tCHSH)：CS# Active Hold Time

c. TSHSL_R：CS #Deselect Time (for Array Read -> ArrayRead)

d. TSHSL_W：CS #Deselect Time (for Erase, Program or Read Status Registers ->Read Status Registers)



C 处标红：当前支持的 flash device 的参数信息填充。（添加需要查看 device 手册如下图位置）。

a. pagesize：device page size without oob size

b. blocksize：device page size without oob size

c. oobsize：oob area size in each page

d. flashsize：chip size without oob size

e. ecc_max：flash 可以检测和纠正的 ECC 最大位数。

f. need_quad：使能四线传输

D 处标红：添加支持的每款 device 信息（按照以下结构体进行填充）。

```
#define DEVICE_ID_STRUCT(id, name_string, parameter)
{\
```

```
    .id_device = id,          \
    .name = name_string,      \
    .param = parameter,       \}
struct device_id_struct {
    uint8_t id_device;                  //device id
    char *name;                         //device name
    struct jz_nand_base_param *param;   //与 C 处标红相对应的参数地址。
};
```

```
static int fs_nand_init(void) {
        struct jz_nand_device *fs_nand;
        fs_nand = kzalloc(sizeof(*fs_nand), GFP_KERNEL);
        if(!fs_nand) {
                pr_err("alloc fs_nand struct fail\n");
                return -ENOMEM;
        }

        fs_nand->id_manufactory = 0xCD;
        fs_nand->id_device_list = device_id;
        fs_nand->id_device_count = FS_DEVICES_NUM;

        fs_nand->ops.nand_read_ops.get_feature = fs_get_read_feature;
        return jz_spinand_register(fs_nand);
}
SPINAND_MOUDLE_INIT(fs_nand_init);
```

A 处标红：填充 flash 的厂商 ID

B 处标红：此处为必填，必须填充***_get_read_feature 函数地址。（主要功能是对 ECC value 进行处理，即***_get_read_feature 函数必须在此文件中实现，具体实现可参考其他 flash 文件的实现）。

C 处标红：填充当前 flash 厂商的特殊用法。主要是将 device 数据手册中的使用法与 nand_common.c 中罗列地进行对比，把特殊用法编写成相应的函数，然后将函数地址填充于此。

步骤 3：修改 Makefile

修改 Makefile 文件，将新添加的 flash 文件添加进去，进行编译。（以 foresee_nand.c 为例）。

```
$ vi drivers/mtd/devices/jz_sfc_v1/Makefile
```

```
LIB := $(obj)libsfc.o

COBJS-$(CONFIG_SFC_NOR) += jz_sfc_nor.o
COBJS-$(CONFIG_MTD_SFCNAND) += jz_sfc_common.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/ato_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/gd_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/mxic_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/nand_common.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/winbond_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/xtx_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/foresee_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=nand_device/dosilicon_nand.o
COBJS-$(CONFIG_MTD_SFCNAND) +=jz_sfc_nand.o
```

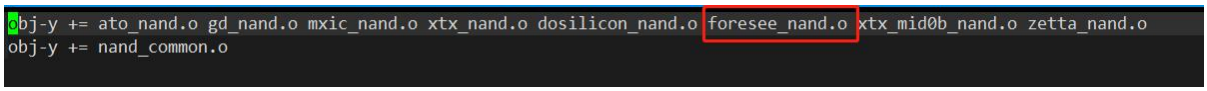
2.3 Kernel 中添加 nand flash 配置

步骤 1：添加配置文件

```
$ cd drivers/mtd/devices/ingenic_sfc_v2/nand_device/
```

如果此目录下没有该厂商 ID 的配置文件，拷贝一份现有的 nand 配置文件进行修改，步骤同 2.2. uboot 中添加 nand flash 配置 步骤 1、步骤 2 类似。

步骤 2：修改 Makefile




```
$ vi drivers/mtd/devices/ingenic_sfc_v2/nand_device/Makefile
```

添加编译此文件（以 foresee_nand.c 为例）。

3 关于单线/四线配置

3.1 uboot 中配置单线/四线

uboot 中默认配置单线模式。

```
$ vi include/configs/isvp_T41.h
```

查看是否开启了 CONFIG_SPI_QUAD 的宏配置，如果没有开启表示默认单线模式，开启表示 uboot 使用四线模式。

3.2 kernel 中配置单线/四线

在 kernel 目录下进行配置：

```
$ make menuconfig
Device Drivers  --->
    <*> Memory Technology Device (MTD) support  --->
        Self-contained MTD device drivers  --->
            <*> Ingenic series SFC driver v2
                sfc Mode (standard spi mode)  --->
                    (X) standard spi mode
                    ( ) quad spi mode
```

单线：standard spi mode

四线：quad spi mode

配置完成可以通过.config 查看：

单线：

```
CONFIG_SPI_STARDARD=y
# CONFIG_SPI_QUAD is not set
```

四线：

```
# CONFIG_SPI_STARDARD is not set

CONFIG_SPI_QUAD=y
```

4 nor 与 nand 的选择及配置

4.1 uboot 中编译 nor 或者 nand

芯片型号	编译命令	说明
T41N	make isvp_t41n_sfc_nor	编译 norflash 启动的 uboot，针对 T41N 芯片
T41L	make isvp_t41l_sfc_nor	编译 norflash 启动的 uboot，针对 T41L 芯片
T41N	make isvp_t41n_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41N 芯片(sfc0 接口)
T41L	make isvp_t41l_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41L 芯片(sfc0 接口)
T41N	make isvp_t41n_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41N 芯片(sfc1 接口)
T41L	make isvp_t41l_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41L 芯片(sfc1 接口)
T41N	make isvp_t41n_msc0	编译 SD 卡启动的 uboot，针对 T41N 芯片(msc0 接口)
T41L	make isvp_t41l_msc0	编译 SD 卡启动的 uboot，针对 T41L 芯片(msc0 接口)
T41N	make isvp_t41n_msc1	编译 SD 卡启动的 uboot，针对 T41N 芯片(msc1 接口)
T41L	make isvp_t41l_msc1	编译 SD 卡启动的 uboot，针对 T41L 芯片(msc1 接口)

注意：SD 卡启动针对 nor flash 还是 nand flash 需要更改以下部分

```
$ vi include/configs/isvp_t41.h
```

```
580 #ifdef CONFIG_SFC_COMMAND/* SD card start */
581 #if 1
582 #define CONFIG_SFC_NOR_COMMAND /* support nor command */
583 #else
584 #define CONFIG_SFC_NAND_COMMAND /* support nand command */
585 #endif
586 #endif /* CONFIG_SFC_COMMAND */
```

该值置 1 时，支持 nor flash 的 sf 命令

该值置 0 时，支持 nand flash 的 nand 命令

4.2 kernel 中配置选择 nor 或者 nand

这里以 T41 为例：

编译选择 NOR ：

```
$ make isvp_marmot_defconfig
```

编译选择 NAND：

```
$ make isvp_marmot_nand_defconfig
```

4.3 uboot 中 32M/64M nor flash 的配置

T41 兼容了 32M/64M 的 nor flash，不再需要添加宏

~~对于 16M 以上的 nor flash，需要在下图位置添加宏 CONFIG_NORFLASH_32M~~

```
~~$ vi include/configs/isvp_T41.h~~
```

5 烧录

对于第一次启动板子，nor flash 中没有数据，需要从 SD 卡启动或者通过 USB 烧录工具（可以参考《USBCloner 烧录指南》），将编译好的 uboot 烧录到 flash 中。 以后就可以直接从 flash 中启动。

5.1 SD 卡制作与烧录

步骤 1：

制作启动卡

格式化 sd 卡， 并将卡启动 uboot 烧录到 SD 卡。

此步骤只需执行一次， 若不更新卡起 uboot 则不需要再次执行此步骤。 经过此步骤处理的 SD 卡可当作正常卡使用。

将 SD 卡插入电脑

注意！！！： 确定插入你的电脑上 TF 卡对应的设备节点文件是不是 sdb(有可能是 sdc 或者 sdd， 可以通过插拔 SD 卡确认)； 如果对应错了可能会破坏电脑的硬盘文件系统。

A. 卸载 SD 卡

```
$ umount /media/user/*
```

B. 将 SD 卡重新分区

```
$ fdisk /dev/sdb
> o --创建一个新的分区表
> n --添加一个分区 n
--此时会提示：
Partition type:
   p primary (0 primary, 0 extended, 4 free)
```

```
    e extended
Select (default p):
Using default response p
--这里回车默认是 p， 创建主分区
分区号 (1-4， 默认为 1)：
将使用默认值 1
--这里回车默认创建 1 号分区
起始 sector (2048-15130623， 默认为 2048)：
将使用默认值 2048
Last sector, +扇区 or +size{K,M,G} (2048-15130623， 默认为 15130623)：
将使用默认值 15130623
--2048 号扇区在 1MB 的位置， 给 uboot 空出了空间
最后一步， 输入 w， 向 SD 卡中写入分区表
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
```

到这里分区表已创建完成；

C. 执行 sync， 拔卡， 重新插卡， PC 将重新识别分区；

D. 格式化 sd 卡为通用的 VFat 文件系统；

E. **编译卡起 u-boot**，选择对应的 uboot 类型(4.1 uboot 编译)，将生成的 u-boot-with-spl.bin 文件烧录进 sd 卡的 17KB 偏移处。(注意是 dev/sdb 分区，不是 dev/sdb1)

```
$ dd if=u-boot-with-spl.bin of=/dev/sdb bs=1024 seek=17
```

步骤 2：

SD 卡启动

bootload 默认最先从读 nor 读取启动信息， 如果 nor 原本烧写过 uboot， 现在想要从卡启， 需要做如下操作：

A. 把制作好的启动卡插入板子上；

B. 长按 boot 键，然后断开电源，上电等 3s，松开 boot 键即可进入卡启 uboot。

步骤 3：

SD 卡烧录（卡启动 uboot 后也可以用 5.2 tftpboot 烧录）

A. 首先，将需要烧录的文件读到内存中，以 u-boot 为例：

B. 把内存先全部清为 f

```
# mw.b 0x80600000 0xff 0x1000000 --第三个参数是大小，这里清了 16MB
```

C. 查看 SD 卡中的文件

```
# fatls mmc 0
    222944 u-boot-with-spl.bin
    2368879 uimage
5 file(s), 0 dir(s)
```

D. load 文件到内存(这里烧录的是 nor/nand 启 uboot)

```
# fatload mmc 0 0x80600000 u-boot-with-spl.bin
```

E. 将内存中的数据写入 nor/nand，写入 nor 和 nand 的命令是不一样的，参考 5.4 sf 与 nand 命令详解

a. 写入 nor（nor 目前可以使用两路 SFC 设备）

```
针对 SFC0_NOR
# sf0 probe  --匹配 flash
# sf0 erase 0x0 0x40000 --擦除 uboot 分区大小 256K（0x40000）
# sf0 write 0x80600000 0x0 0x40000 --将 uboot 写入 Nor 0x0~0x40000 处

针对 SFC1_NOR
# sf1 probe  --匹配 flash
# sf1 erase 0x0 0x40000 --擦除 uboot 分区大小 256K（0x40000）
# sf1 write 0x80600000 0x0 0x40000 --将 uboot 写入 Nor 0x0~0x40000 处
```

b. 写入 nand（nand 目前只能使用一路）

```
# nand erase 0x0 0x100000 --擦除 uboot 分区大小 1M
# nand write 0x80600000 0x0 0x100000 --将 uboot 写入 Nor 0x0~0x100000 处
```

F. 此时已经把 uboot 烧录到 flash 中，下次启动可以直接从 flash 中的 uboot 启动。

5.2 tftpboot 烧录

将文件通过 tftp 方式， 从 PC 端， 下载到 Uboot 的内存中， 然后写到 flash 中。

步骤 1：

硬件：开发板上网卡；板子通过网线连接到路由器或者交换机上，PC 也连到该路由器或者交换机上，并且处于同一网段上；

软件：PC 端安装并设置好 TFTP 服务，把相应的 u-boot 等文件放到 TFTP 根目录下；uboot 中支持网卡驱动并且支持相应的 TFTP 操作。

在 uboot 中， 执行 `$ tftpboot mem_addr file_name` ； 可以将文件 file_name 传送到 Uboot 的内存地址 mem_addr 中。

步骤 2：

烧录到 Nor

通过 tftp 命令 load 文件到内存；

```
# mw.b 0x80600000 0xff 0x1000000 --清 16MB 内存
# tftpboot 0x80600000 u-boot-with-spl.bin
# tftpboot 0x80640000 uImage
# tftpboot 0x808c0000 root-uclibc-toolchain720.squashfs
```

分别把 u-boot-with-spl.bin、 uImage、 root-uclibc-toolchain720.squashfs 下载到内存的 0x806000000、 0x80640000、 0x808c0000 处。

把下载到内存上的文件， 写到开发板上的 nor flash 上， 重启后可以直接从 nor 上读取文件， 而不必每次手动 load 文件到内存上。 烧录命令如下：

```
# sf0 probe;sf0 erase 0x0 0x1000000;sf0 write 0x80600000 0x0 0x1000000
```

或者

```
# sf1 probe;sf1 erase 0x0 0x1000000;sf1 write 0x80600000 0x0 0x1000000
```

参考 5.4 sf 与 nand 命令详解。

烧录到 Nand

通过 tftp 命令 load 文件到内存；

```
# mw.b 0x80600000 0xff 0x1800000 --清 24MB 内存
```

```
# tftpboot 0x80600000 u-boot-with-spl.bin
# tftpboot 0x80700000 uImage
# tftpboot 0x80a00000 rootfs720.img
```

分别把 u-boot-with-spl.bin、uImage、rootfs720.img 下载到内存的 0x806000000、0x80700000、0x80a00000 处。

把下载到内存上的文件，写到开发板上的 nand flash 上，烧录命令如下：

```
# nand erase 0x0 0x8000000;nand write 0x80600000 0x0 0x8000000
```

参考 5.4 解释

5.3 USB 烧录

参考《USBCloner 烧录指南.pdf》。

5.4 sf 与 nand 命令详解

对于 SFC 模块，T40 只有一路 SFC，T41 有两路 SFC，也就是说 T41 可以挂载两块 flash 设备，**故在烧录命令上也有改动.**

1）对于 nor（可以同时挂载两块 nor flash）

针对 SFC0_NOR （注：sf 命令和 sf0 效果是一样的）

```
# sf0 probe  --匹配 flash 信息
# sf0 erase <flash_addr> <size> --擦除 flash，例如：
# sf0 erase 0x0 0x1000000
```

从 nor flash 的 0x0 位置开始擦除，擦除大小为 0x1000000（16M）

```
# sf0 write <ddr_addr> <flash_addr> <size> --写 flash，例如：
# sf0 write 0x80600000 0x0 0x1000000
```

从 ddr 地址 0x80600000 开始，将大小为 0x1000000 的内容写入到 nor flash 的 0x0 位置，写入的大小为 0x1000000。

针对 SFC1_NOR

只需将 sf0 替换成 sf1 即可，解释同上

2）对于 nand（nand 目前只能同时挂载一块）

备注：nand 命令不需要执行 probe

```
# nand erase <flash_addr> <size> --擦除 flash，例如
# nand erase 0x0 0x100000
从 nand flash 的 0x0 位置开始擦除，擦除大小为 0x100000（1M）

# nand write <ddr_addr> <flash_addr> <size> --写 flash，例如：
# nand write 0x80600000 0x0 0x100000
从 ddr 地址 0x80600000 开始，将大小为 0x100000 的内容写入到 nor flash 的 0x0 位置，写入的大小为 0x100000。
```