1. MTD 시스템

임베디드 리눅스 장비에서 램디스크를 이용하여 루트 파일 시스템을 구현 하였을 경우에는 보드 동작 중에 파일로 기록된 내용이 전원이 꺼짐과 동시에 소실된다. 기록된 내용을 영구 저장 하기 위해서는 일반적으로 플래시 메모리에 기록하여야 한다. 플래시 메모리를 리눅스의 루트 파일 시스템으로 사용하기 위해서는 MTD (Memory Technology Device) 블록 디바이스 드라이버를 사용하여야 한다.

타겟 보드는 NAND 플래시 기반의 보드이다. 보조 기억장치로써 NOR 형 플래시를 사용하지 않고 NAND 형 플래시를 사용한다.

타겟 보드에서 NAND 기반의 플래시를 사용하게 된 이유는 기존 이지보드의 영향이 매우 크다.

첫번째, 보드의 생산량이 많아질 경우 민감해지는 보드 단가 부분에서 NAND 형 플래시보다 NOR형 플래시가 월등히 비싸다.

두번째, 쓰기 속도의 문제이다. NOR형 플래시의 읽기 속도는 일반 ROM의 속도로 사용되어 이익을 얻을 수 있지만 쓰기 속도는 너무 느리게 동작한다. 또 NAND에 비해서 읽기 속도도 전반적인 속도 평균을 보면 그리 빠른 편이라 볼 수 없다.

세번째, 현장에서 실제 파일 입출력을 사용 했을 때 의외로 NOR 플래시 시스템에 문제가 발생함을 알 수 있다.

그래서 본사의 개발팀은 여러 난상 토론 끝에 NAND 플래시 시스템을 쓰기로 결정했다.

이 NAND 플래시 시스템을 파일 시스템으로 사용하기 위해서는 다음의 두가지가 있다

- MTD + JFFS2 방식
- MTD + YAFFS 방식

이중에서 자체 실험결과 NAND 시스템에서는 JFFS2 방식이 무척 불합리하다는 것으로 판단되어 최종적으로 YAFFS 시스템으로 결정이 되었다.

아직 완전한 검증이 되지 않은 시스템이라는 점이 문제이기는 하나 본사의 자체 시험결과 문제점 발생을 찾지 못했다.

굳이 문제점을 지적하라면 동일 크기의 데이터를 기록함에 있어서 사용량이 JFFS2 보다 많이 소모된다.

그러나 이점은 읽기 속도와 쓰기 속도의 향상으로 충분히 보상되고도 남으며 또한 NAND의 단가가 매우 낮기 때문에 별 문제가 되지 않을 것으로 본다.

현재 제공되고 있는 커널 소스는 MTD와 JFFS2 그리고 YAFFS에 포함되어 있다. 그러나 본 문서의 커널 패치에서는 이 부분이 빠져 있는데 이유는 그 과정이 복잡하기 때문이다.
그래서 따로 이렇게 한 장을 할애 하여 커널에 NAND시스템을 위한 MTD 시스템을 구현에 대하여 설명하고 있다.

우선 진행하기 이전에 MTD에 대해서 간단하게 살펴 보고자 한다.

■ **MTD**

MTD는 디바이스 드라이버이다. 이 디바이스 드라이버가 지원하려 했던 초기 모델은 DOC(Disk On Chip) 이다. 그 후 임베디드 시스템의 플래시를 지원했으며 현재의 형태가 되었다.

MTD 관련 공식 사이트는 http://www.linux-mtd.infradead.org/ 이다
다음으로 접속하면 mtd관련 파일을 얻을 수 있다.

ftp://ftp.uk.linux.org/pub/people/dwmw2/mtd/cvs/

MTD가 지원하는 플래시는 여러 형태가 있는데 그 중 우리의 관심은 NAND 이다.
기존 ARM을 지원하는 커널 2.6.8-rc2에서는 이 MTD에 대한 내용이 포함되어 있다. 따라서 이것을 그대로 사용하기로 한다.

MTD는 두 가지 형태의 디바이스 드라이버를 지원하는데 캐릭터형과 블럭형 이다.
캐릭터형은 플래시자체를 접근 가능하게 하는 부분이 구현되어 있다. 그러므로 플래시를 직접 지우거나 할 경우에는 캐릭터형으로 접근하여야 한다.

장치 파일을 만들때 주번호는 90으로 시작하며 짝수의 부 번호는 읽기 쓰기가 가능하고 홀수의 번호는 읽기만 가능하다. 이렇게 구별한 이유는 아마도 플래시를 보호하기 위한 것으로 보인다. 그러나 타겟 보드에서 램디스크에 미리 만들어 놓은 것은 읽기 쓰기가 가능한 짝수 부 번호이다.

다음은 이에 대한 내용을 보여주는 화면이다.

```
[root@falinux dev]$ ls -al mtd?
crw-r--r--    1 root     root      90,   0 Mar  2  2002 mtd0
crw-r--r--    1 root     root      90,   2 Mar  2  2002 mtd1
crw-r--r--    1 root     root      90,   4 Mar  2  2002 mtd2
crw-r--r--    1 root     root      90,   6 Mar  2  2002 mtd3
crw-r--r--    1 root     root      90,   8 Mar  2  2002 mtd4
crw-r--r--    1 root     root      90,  10 Mar  2  2002 mtd5
crw-r--r--    1 root     root      90,  12 Mar  2  2002 mtd6
crw-r--r--    1 root     root      90,  14 Mar  2  2002 mtd7
crw-r--r--    1 root     root      90,  16 Mar  2  2002 mtd8
crw-r--r--    1 root     root      90,  18 Mar  2  2002 mtd9
[root@falinux dev]$ ls -al mtd1?
crw-r--r--    1 root     root      90,  20 Mar  2  2002 mtd10
crw-r--r--    1 root     root      90,  22 Mar  2  2002 mtd11
crw-r--r--    1 root     root      90,  24 Mar  2  2002 mtd12
crw-r--r--    1 root     root      90,  26 Mar  2  2002 mtd13
crw-r--r--    1 root     root      90,  28 Mar  2  2002 mtd14
crw-r--r--    1 root     root      90,  30 Mar  2  2002 mtd15
[root@falinux dev]$
```

블럭형은 플래시를 파일 시스템에서 접근하게 한 것이다. 당연히 이 장치파일은 응용 프로그램에서 접근하면 곤란하다. 특별한 경우가 아니면 접근하지 말아야 한다. 블럭형 장치파일의 일반적인 사용은 mount 명령을 사용할 때 사용된다.

다음은 이에 대한 내용을 보여주는 화면이다.

```
[root@falinux dev]$ ls -al mtdblock?
brw-r--r--    1 root     root      31,   0 Jan 12  2002 mtdblock0
brw-r--r--    1 root     root      31,   1 Jan 12  2002 mtdblock1
brw-r--r--    1 root     root      31,   2 Jan 12  2002 mtdblock2
brw-r--r--    1 root     root      31,   3 Jan 12  2002 mtdblock3
brw-r--r--    1 root     root      31,   4 Jan 12  2002 mtdblock4
brw-r--r--    1 root     root      31,   5 Jan 12  2002 mtdblock5
brw-r--r--    1 root     root      31,   6 Jan 12  2002 mtdblock6
brw-r--r--    1 root     root      31,   7 Jan 12  2002 mtdblock7
brw-r--r--    1 root     root      31,   8 Jan 12  2002 mtdblock8
brw-r--r--    1 root     root      31,   9 Jan 12  2002 mtdblock9
[root@falinux dev]$ ls -al mtdblock1?
brw-r--r--    1 root     root      31,  10 Jan 12  2002 mtdblock10
brw-r--r--    1 root     root      31,  11 Jan 12  2002 mtdblock11
brw-r--r--    1 root     root      31,  12 Jan 12  2002 mtdblock12
brw-r--r--    1 root     root      31,  13 Jan 12  2002 mtdblock13
brw-r--r--    1 root     root      31,  14 Jan 12  2002 mtdblock14
brw-r--r--    1 root     root      31,  15 Jan 12  2002 mtdblock15
brw-r--r--    1 root     root      31,  16 Jan 12  2002 mtdblock16
[root@falinux dev]$
```

1.1. MTD 에 NAND 지원 패치

MTD 최신본(?)을 커널에 패치 했다면 다음에 NAND를 지원하게 MTD를 수정하여야 한다.

NAND 의 종류가 몇 가지 안되고 자랑스럽게도 한국의 삼성 NAND가 전세계를 지배했어도 NAND의 컨트롤 방식은 보드마다 다르다.
그러므로 이에 대한 패치를 수행하여야 한다.

또 한가지는 NAND의 ECC 문제 때문에 끊임없이 경고 메세지를 발생하는 문제가 있는데 이를 수정하여야 한다.

./drivers/mtd/nand/nand_base.c 를 수정한다.

printk 문을 주석 처리를 하면 된다.

nand_write_page 함수 중에서 다음 라인을 주석 처리한다.
[수정부분 #1]

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

```
811         switch (eccmode) {
812         /* No ecc, write all */
813         case NAND_ECC_NONE:
814                 // printk (KERN_WARNING "Writing data without ECC to NAND-FLASH is not recommended\n");
815                 this->write_buf(mtd, this->data_poi, mtd->oobblock);
816                 break;
```

nand_read_ecc 함수 중에서 다음 라인을 주석 처리한다.
[수정부분 #2]

// printk (KERN_WARNING "Reading data from NAND FLASH without ECC is not recommended\n");

```
1136                switch (eccmode) {
1137                case NAND_ECC_NONE: {   /* No ECC, Read in a page */
1138                        static unsigned long lastwhinge = 0;
1139                        if ((lastwhinge / HZ) != (jiffies / HZ)) {
1140                        //      printk (KERN_WARNING "Reading data from NAND FLASH without ECC is not recommended\n");
1141                                lastwhinge = jiffies;
1142                        }
```

nand_scan 함수 중에서 다음 라인을 주석 처리한다.
[수정부분 #3]

// printk (KERN_WARNING "NAND_ECC_NONE selected by board driver. This is not recommended !!\n");

```
2472        case NAND_ECC_NONE:
2473                //printk (KERN_WARNING "NAND_ECC_NONE selected by board driver. This is not recommended !!\n");
2474                this->eccmode = NAND_ECC_NONE;
2475                break;
2476
2477        case NAND_ECC_SOFT:
```

./drivers/mtd/nand/ez_s2410.c 작성

타겟 보드에 추가되는 nand를 지원하기 위한 루틴을 작성해야 하는데 소스는 제공된 CD의 /sw/mtd/ez_s2410.c 에 있으며, 내용을 작성하지 않고 복사하여 사용하면 된다.

다음은 소스의 내용이다.

```
/*
 *  drivers/mtd/nand/ez_s2410.c
 *
 *  Copyright (C) 2003 You youngchang (frog@falinux.com)
 *
 *  Support board - EZ-S2410
 */

#include <linux/slab.h>
#include <linux/module.h>
#include <linux/mtd/mtd.h>
#include <linux/mtd/nand.h>
#include <linux/mtd/partitions.h>
#include <linux/string.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/compiler.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include <linux/init.h>
#include <linux/pci.h>
#include <linux/delay.h>
#include <linux/ethtool.h>
#include <linux/mii.h>
#include <linux/if_vlan.h>
#include <linux/crc32.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <asm/io.h>
#include <asm/uaccess.h>

#define EZ_NAND_BASE            S3C2410_VA_NAND
#define EZ_NAND_CTRL            (EZ_NAND_BASE+0x0)
#define EZ_NAND_DATA            (EZ_NAND_BASE+0xC)
#define EZ_NAND_CMD             (EZ_NAND_BASE+0x4)
#define EZ_NAND_ADDR            (EZ_NAND_BASE+0x8)
#define EZ_NAND_STATE           (EZ_NAND_BASE+0x10)

#define EZ_NAND_nFCE            (1<<11)

static struct mtd_info *ez_s2410_nand_mtd = NULL;     // 정보 테이블

#define SZ_1M     (1024*1024)
#define SZ_1K     (1024)

static char *cmdline_par;
```

```
/*
 * Define partitions for flash device
 */
#ifdef CONFIG_MTD_PARTITIONS
static struct mtd_partition partition_info[] =
{
        {
                .name   = "mtdblock0 Kernel partition",
                .offset = 0x40000,
                .size   = (2*SZ_1M) - 0x40000,
        },
        {
                .name   = "mtdblock1 Ramdisk partition",
                .offset = 0x400000,
                .size   = 5*SZ_1M
        },
        {
                .name   = "mtdblock2(yaffs) Data partition 0",
                .offset = 0x700000,
                .size   = 57*SZ_1M
        }
};
#define EZ_S2410_NAND_NUM_PARTITIONS    3
#endif

// 커널 커맨드라인을 파싱한다.
static void fixup_partition_info( void )
{
        char *delim_ = ",";
        int  argc;
        char *argv[256];
        char *tok;
        int  size[3];

        argc       = 0;
        argv[argc] = NULL;
        for (tok = strsep( &cmdline_par, delim_); tok; tok = strsep( &cmdline_par, delim_))
        {
                argv[argc++] = tok;
        }

        if ( argc == EZ_S2410_NAND_NUM_PARTITIONS )
        {
                size[0] = simple_strtoul( argv[0],NULL,0 );
                size[1] = simple_strtoul( argv[1],NULL,0 );
                size[2] = simple_strtoul( argv[2],NULL,0 );

                if ( ( size[0] > 0 ) && ( size[1] > 0 ) && ( size[2] > 0 ) )
                {
                        partition_info[0].offset =  0x40000;
                        partition_info[0].size   = (size[0]*SZ_1M) - partition_info[0].offset;

                        partition_info[1].offset =  size[0]*SZ_1M;
                        partition_info[1].size   =  size[1]*SZ_1M;

                        partition_info[2].offset = (size[0]+size[1])*SZ_1M;
                        partition_info[2].size   =  size[2]*SZ_1M;
                }
        }
}
```

```
/*
 *      hardware specific access to control-lines
*/
void ez_s2410_nand0_hwcontrol(struct mtd_info *mtd, int cmd)
{
    unsigned short dummy;

    switch(cmd)
    {
        case NAND_CTL_SETNCE: writew( readw(EZ_NAND_CTRL) & (~EZ_NAND_nFCE), EZ_NAND_CTRL ); break;
        case NAND_CTL_CLRNCE: writew( readw(EZ_NAND_CTRL) |   EZ_NAND_nFCE , EZ_NAND_CTRL ); break;
    }
}

/*
 * Send command to NAND device
 */
void ez_s2410_nand_command (struct mtd_info *mtd, unsigned command, int column, int page_addr)
{
        register struct nand_chip *this = mtd->priv;
        register unsigned long NAND_IO_ADDR = this->IO_ADDR_W;

        // Write out the command to the device.
        if (command != NAND_CMD_SEQIN)
        {
                writew (command, EZ_NAND_CMD );
        }
        else
        {
                if (mtd->oobblock == 256 && column >= 256)
                {
                        column -= 256;
                        writew (NAND_CMD_READOOB, EZ_NAND_CMD );
                        writew (NAND_CMD_SEQIN  , EZ_NAND_CMD );
                }
                else if (mtd->oobblock == 512 && column >= 256)
                {
                        if (column < 512)
                        {
                                column -= 256;
                                writew (NAND_CMD_READ1, EZ_NAND_CMD);
                                writew (NAND_CMD_SEQIN, EZ_NAND_CMD);
                        }
                        else
                        {
                                column -= 512;
                                writew (NAND_CMD_READOOB, EZ_NAND_CMD);
                                writew (NAND_CMD_SEQIN  , EZ_NAND_CMD);
                        }
                }
                else
                {
                        writew (NAND_CMD_READO      , EZ_NAND_CMD);
                        writew (NAND_CMD_SEQIN      , EZ_NAND_CMD);
                }
        }

        // Serially input address
        if (column != -1 || page_addr != -1)
        {
            if (column != -1) writeb (column, EZ_NAND_ADDR);
            if (page_addr != -1)
            {
                writew ((unsigned char) (page_addr & 0xff), EZ_NAND_ADDR);
```

```
                writew ((unsigned char) ((page_addr >> 8) & 0xff), EZ_NAND_ADDR);
                // One more address cycle for higher density devices
                if (mtd->size & 0x0c000000)
                {
                    writew ((unsigned char) ((page_addr >> 16) & 0x0f), EZ_NAND_ADDR);
                }
            }
        }

        switch (command)
        {
        case NAND_CMD_PAGEPROG:
        case NAND_CMD_ERASE1:
        case NAND_CMD_ERASE2:
        case NAND_CMD_SEQIN:
        case NAND_CMD_STATUS:
                    return;

        case NAND_CMD_RESET:
                if( this->dev_ready )   break;
                writeb (NAND_CMD_STATUS, EZ_NAND_CMD);
                while ( !(readb (EZ_NAND_STATE) & 0x1));
                return;

        default:
                if (!this->dev_ready)
                {
                        udelay (this->chip_delay);
                        return;
                }
        }
        while (!this->dev_ready(mtd));
}

int ez_s2410_device_ready(struct mtd_info *mtd)
{
        return (readb (EZ_NAND_STATE) & 0x1) ? 1 : 0;
}

/*
 * Main initialization routine
 */
//int __init ez_s2410_nand_init (void)
static int __init ez_s2410_nand_init (void)
{
        struct nand_chip *this;

        //  Allocate memory for MTD device structure and private data
        ez_s2410_nand_mtd = kmalloc (sizeof(struct mtd_info) + sizeof (struct nand_chip), GFP_KERNEL);
        if (!ez_s2410_nand_mtd)
        {
                printk ("Unable to allocate EZ-S2410-NAND MTD device structure.₩n");
                return -ENOMEM;
        }

        // Get pointer to private data
        this = (struct nand_chip *) (&ez_s2410_nand_mtd[1]);

        // Initialize structures
        memset((char *) ez_s2410_nand_mtd, 0, sizeof(struct mtd_info));
        memset((char *) this, 0, sizeof(struct nand_chip));

        // Link the private data with the MTD structure
        ez_s2410_nand_mtd->priv = this;
```

```
        // Set address of NAND IO lines
        this->IO_ADDR_R = EZ_NAND_DATA;
        this->IO_ADDR_W = EZ_NAND_DATA;
        // Set address of hardware control function
        this->hwcontrol = ez_s2410_nand0_hwcontrol;
        this->dev_ready = ez_s2410_device_ready;
        // Set commamd function
        this->cmdfunc = ez_s2410_nand_command ;
        // 15 us command delay time
        this->chip_delay = 15;

        this->eccmode = NAND_ECC_SOFT;
        ez_s2410_nand_mtd->oobsize = 16;

        // Scan to find existence of the device
        if (nand_scan (ez_s2410_nand_mtd,1))
        {
                kfree (ez_s2410_nand_mtd);
                return -ENXIO;
        }

        // 커널커맨드에서 정보를 얻는다
        fixup_partition_info();

        // Register the partitions
        add_mtd_partitions(ez_s2410_nand_mtd, partition_info, EZ_S2410_NAND_NUM_PARTITIONS);

        // Return happy
        return 0;
}

module_init(ez_s2410_nand_init);

/*
 * Clean up routine
 */
#ifdef MODULE
static void __exit ez_s2410_nand_cleanup (void)
{
        struct nand_chip *this = (struct nand_chip *) &ez_s2410_nand_mtd[0];

        // Unregister the device
        del_mtd_device (ez_s2410_nand_mtd);

        // Free internal data buffer
        kfree (this->data_buf);

        // Free the MTD device structure
        kfree (ez_s2410_nand_mtd);
}
module_exit(ez_s2410_nand_cleanup);
#endif

static int __init nandpart_setup(char *s)
{
        cmdline_par = s;
        return 1;
}

__setup("nandparts=", nandpart_setup);

MODULE_AUTHOR("You Youngchang, Jang Hyung-Gi <frog@falinux.com");
MODULE_DESCRIPTION("Board-specific glue layer for NAND flash on EZ-S2410-NAND board");
MODULE_LICENSE("GPL");
```

./drivers/mtd/nand/Kconfig 파일에 다음 내용을 추가 한다.

```
config MTD_NAND_EZ_S2410
        tristate "NAND Flash device for S3C2410 on EZ-S2410 board"
        depends on ARM && ARCH_S3C2410 && MTD_NAND
        help
          If you had to ask, you don't have one. Say 'N'.
```

[수정전]

```
15 config MTD_NAND_VERIFY_WRITE
16         bool "Verify NAND page writes"
17         depends on MTD_NAND
18         help
19           This adds an extra check when data is written to the flash. The
20           NAND flash device internally checks only bits transitioning
21           from 1 to 0. There is a rare possibility that even though the
22           device thinks the write was successful, a bit could have been
23           flipped accidentaly due to device wear or something else.
24
25 config MTD_NAND_AUTCPU12
26         tristate "SmartMediaCard on autronix autcpu12 board"
27         depends on ARM && MTD_NAND && ARCH_AUTCPU12
28         help
29           This enables the driver for the autronix autcpu12 board to
30           access the SmartMediaCard.
31
32 config MTD_NAND_EDB7312
```

[수정후]

```
config MTD_NAND_VERIFY_WRITE
        bool "Verify NAND page writes"
        depends on MTD_NAND
        help
          This adds an extra check when data is written to the flash. The
          NAND flash device internally checks only bits transitioning
          from 1 to 0. There is a rare possibility that even though the
          device thinks the write was successful, a bit could have been
          flipped accidentaly due to device wear or something else.

config MTD_NAND_EZ_S2410
        tristate "NAND Flash device for S3C2410 on EZ-S2410 board"
        depends on ARM && ARCH_S3C2410 && MTD_NAND
        help
          If you had to ask, you don't have one. Say 'N'.

config MTD_NAND_AUTCPU12
        tristate "SmartMediaCard on autronix autcpu12 board"
        depends on ARM && MTD_NAND && ARCH_AUTCPU12
```

./drivers/mtd/nand/Makefile 수정

타겟 보드를 지원하기 위하여 항목을 추가한다.

obj-$(CONFIG_MTD_NAND_EZ_S2410) += ez_s2410.o

[수정전]

```
#
# linux/drivers/nand/Makefile
#
# $Id: Makefile.common,v 1.9 2004/07/12 16:07:31 dwmw2 Exp $

obj-$(CONFIG_MTD_NAND)                  += nand.o nand_ecc.o
obj-$(CONFIG_MTD_NAND_IDS)              += nand_ids.o

obj-$(CONFIG_MTD_NAND_SPIA)             += spia.o
obj-$(CONFIG_MTD_NAND_TOTO)             += toto.o
obj-$(CONFIG_MTD_NAND_AUTCPU12)         += autcpu12.o
obj-$(CONFIG_MTD_NAND_EDB7312)          += edb7312.o
obj-$(CONFIG_MTD_NAND_TX4925NDFMC)      += tx4925ndfmc.o
obj-$(CONFIG_MTD_NAND_TX4938NDFMC)      += tx4938ndfmc.o
obj-$(CONFIG_MTD_NAND_AU1550)           += au1550nd.o
obj-$(CONFIG_MTD_NAND_PPCHAMELEONEVB)   += ppchameleonevb.o
obj-$(CONFIG_MTD_NAND_DISKONCHIP)       += diskonchip.o

nand-objs = nand_base.o nand_bbt.o
```

[수정후]

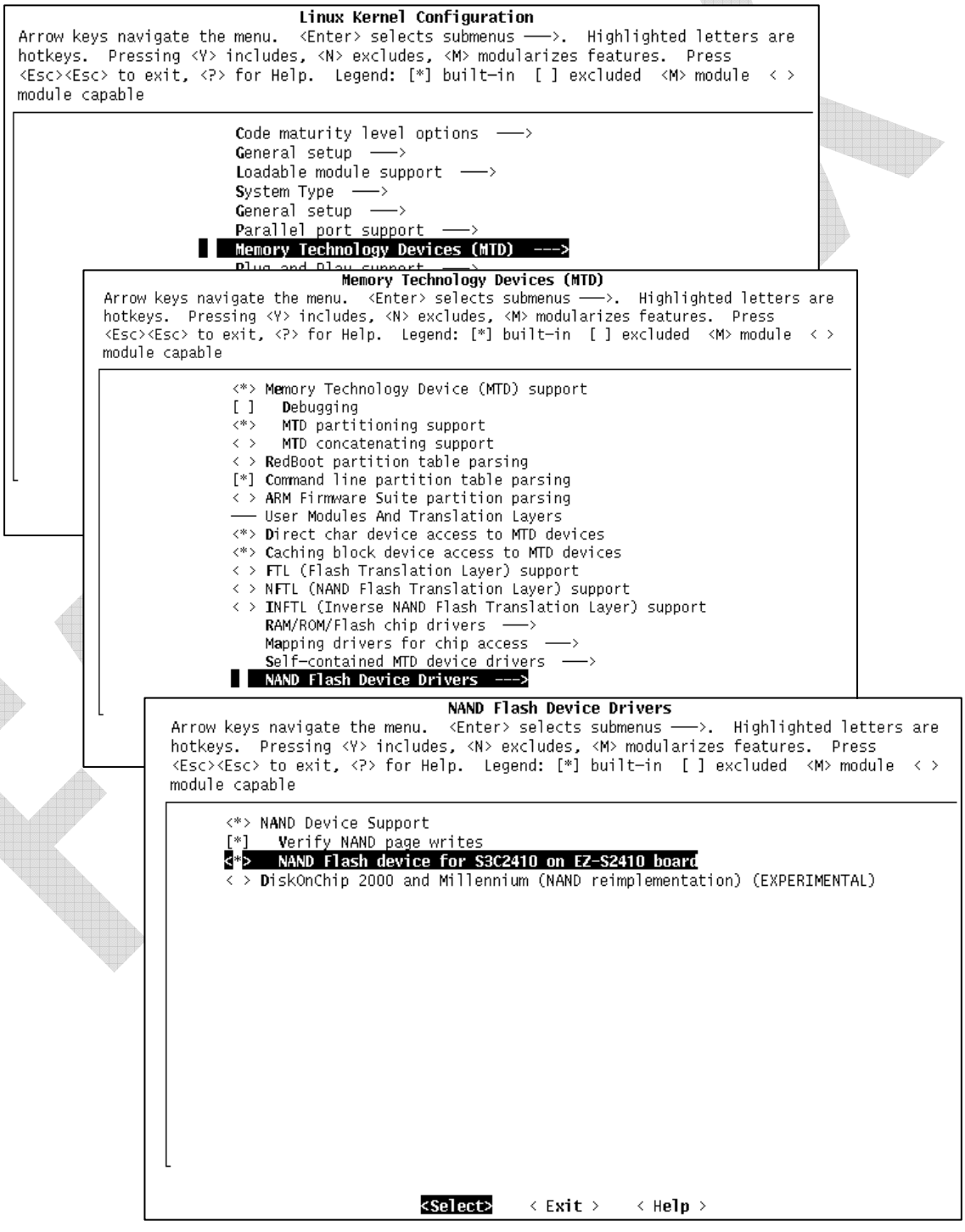
```
#
# linux/drivers/nand/Makefile
#
# $Id: Makefile.common,v 1.9 2004/07/12 16:07:31 dwmw2 Exp $

obj-$(CONFIG_MTD_NAND)                  += nand.o nand_ecc.o
obj-$(CONFIG_MTD_NAND_IDS)              += nand_ids.o

obj-$(CONFIG_MTD_NAND_SPIA)             += spia.o
obj-$(CONFIG_MTD_NAND_TOTO)             += toto.o
obj-$(CONFIG_MTD_NAND_AUTCPU12)         += autcpu12.o
obj-$(CONFIG_MTD_NAND_EDB7312)          += edb7312.o
obj-$(CONFIG_MTD_NAND_TX4925NDFMC)      += tx4925ndfmc.o
obj-$(CONFIG_MTD_NAND_TX4938NDFMC)      += tx4938ndfmc.o
obj-$(CONFIG_MTD_NAND_AU1550)           += au1550nd.o
obj-$(CONFIG_MTD_NAND_PPCHAMELEONEVB)   += ppchameleonevb.o
obj-$(CONFIG_MTD_NAND_DISKONCHIP)       += diskonchip.o
obj-$(CONFIG_MTD_NAND_EZ_S2410)            += ez_s2410.o

nand-objs = nand_base.o nand_bbt.o
```

1.2. 커널 옵션 및 커널 패치

모든 내용을 수정했다면 커널 옵션에 MTD에 관련된 것을 설정해야 한다.

다음은 설정에 대한 화면이다.

 # make menuconfig



위와 같이 설정이 끝났다면 저장한다.

이제 커널을 컴파일 한다.

 # make dep
 # make clean
 # make zImage

타겟 보드에 커널을 다운로딩하고 부팅한다.

이때 부팅 중에 다음과 같은 파티션 정보 메세지가 나오면 정상이다.

```
eth0: cs8900 rev K found at 0xf1000300 [Cirrus EEPROM]
cs89x0: No EEPROM, relying on command line....
cs89x0 media RJ-45, IRQ 37, programmed I/O, MAC 00:a2:55:f2:26:35
cs89x0_probe1() successful
cs89x0: no cs8900 or cs8920 detected.  Be sure to disable PnP with SETUP
NAND device: Manufacturer ID: 0xec,Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 1293 at 0x01434000
Bad eraseblock 1896 at 0x01da0000
Creating 3 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00040000-0x00200000 : "mtdblock0 Kernel partition"
0x00200000-0x00700000 : "mtdblock1 Ramdisk partition"
0x00700000-0x04000000 : "mtdblock2(yaffs) Data partition 0"
s3c2410-ohci s3c2410-ohci0: new USB bus registered, assigned bus number 1
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new driver usb-storage
USB Mass Storage support registered.
usbcore: registered new driver usbhid
```

타겟 보드는 아래와 같이 3개의 파티션으로 나누어져 있다.

/dev/mtd0, /dev/mtdblock0 : 커널 이미지 파티션
/dev/mtd1, /dev/mtdblock1 : 램 디스크 이미지 파티션
/dev/mtd2, /dev/mtdblock2 : 일반 영역 파티션

이중에 응용 프로그램에서 사용할 수 있는 것은 다음과 같다.

- /dev/mtd2
- /dev/mtdblock2

일단 정상적인지 시험하기 위해서 일반 영역을 지워보자

타겟에 넣어져 있는 램디스크 이미지 안에는 **eraseall** 이라는 MTD 의 삭제 유틸리티가 있다.

$ ./eraseall /dev/mtd2

다음은 타겟 보드에서의 실행 화면이다.

```
[root@falinux ~]$ cd /sbin/
[root@falinux sbin]$ df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                11895      7317      3964  65% /
/dev/mtdblock2           58352       128     58224   0% /app
[root@falinux sbin]$ umount /app/
[root@falinux sbin]$ df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                11895      7317      3964  65% /
[root@falinux sbin]$ ./eraseall /dev/mtd2
Erasingnand_erase: attempt to erase a bad block at page 0x0000a1a0
Erasing 16 Kibyte @ d34000 -- 23 % complete.
Erased 58368 Kibyte @ 0 -- 100% complete.
[root@falinux sbin]$
```

이 유틸리티를 사용하지 않고 부트로드에서 삭제할 수도 있다. 위의 유틸리티를 사용할 경우에는 디폴트로 마운트 되어 있는 것을 해지하고 지워야 한다. 따라서 부트로드에서 NAND 파티션을 지우는 것을 추천한다.