1. 디바이스 드라이버 [GPIO]

1.1. GPIO 란

GPIO(General Purpose Input/Output)란 일반적인 용도로 사용 가능한 디지털 입출력 기능의 Port pins 이다. S3C2410의 GPIO는 총 117개이며 각각이 pin 들은 input/output으로 프로그램 되거나 인터럽트 source로 사용될 수 있다.

S3C2410의 대부분의 GPIO는 단순히 디지털 입출력 뿐만 아니라 부가적인 기능을 갖고 있다. 그래서 다른 기능을 사용하다 보면 처음 생각보다 단순 IO로 사용할 GPIO가 적어짐을 알 수 있다. 따라서 하드웨어 설계자는 GPIO를 사용할 때 이점을 주의해야 한다.

출력에 관계된 정확한 사양은 레퍼런스 매뉴얼의 하드웨어 특성에 관련된 사항을 참조하기 바란다.

1.2. GPIO 레지스터 설명

S3C2410의 GPIO Port 는 다음과 같다.

- Port A (GPA) : 23-output port
- Port B (GPB) : 11-input/output port
- Port C (GPC) : 16-input/output port
- Port D (GPD) : 16-input/output port
- Port E (GPE) : 16-input/output port
- Port F (GPF) : 8-input/output port
- Port G (GPG) : 16-input/output port
- Port H (GPH) : 11-input/output port

정확한 레지스터의 이름과 주소는 다음과 같다.

레지스터 주소	이름	접근	설 명
0x56000000	GPACON	읽기/쓰기	GPA[22..0] 출력/부가기능 사용 허가 설정 레지스터
0x56000004	GPADAT	읽기/쓰기	GPA[22..0] DATA 출력 레지스터
0x56000010	GPBCON	읽기/쓰기	GPB[10..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000014	GPBDAT	읽기/쓰기	GPB[10..0] DATA 입력/출력 레지스터
0x56000018	GPBUP	읽기/쓰기	GPB[10..0] Pull-up disable 레지스터
0x56000020	GPBCON	읽기/쓰기	GPC[15..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000024	GPBDAT	읽기/쓰기	GPC[15..0] DATA 입력/출력 레지스터
0x56000028	GPBUP	읽기/쓰기	GPC[15..0] Pull-up disable 레지스터
0x56000030	GPBCON	읽기/쓰기	GPD[15..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000034	GPBDAT	읽기/쓰기	GPD[15..0] DATA 입력/출력 레지스터
0x56000038	GPBUP	읽기/쓰기	GPD[15..0] Pull-up disable 레지스터
0x56000040	GPBCON	읽기/쓰기	GPE[15..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000044	GPBDAT	읽기/쓰기	GPE[15..0] DATA 입력/출력 레지스터
0x56000048	GPBUP	읽기/쓰기	GPE[15..0] Pull-up disable 레지스터
0x56000050	GPBCON	읽기/쓰기	GPF[7..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000054	GPBDAT	읽기/쓰기	GPF[7..0] DATA 입력/출력 레지스터
0x56000058	GPBUP	읽기/쓰기	GPF[7..0] Pull-up disable 레지스터
0x56000060	GPBCON	읽기/쓰기	GPG[15..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000064	GPBDAT	읽기/쓰기	GPG[15..0] DATA 입력/출력 레지스터
0x56000068	GPBUP	읽기/쓰기	GPG[15..0] Pull-up disable 레지스터
0x56000070	GPBCON	읽기/쓰기	GPH[10..0] 입력/출력/부가기능 사용 허가 설정 레지스터
0x56000074	GPBDAT	읽기/쓰기	GPH[10..0] DATA 입력/출력 레지스터
0x56000078	GPBUP	읽기/쓰기	GPH[10..0] Pull-up disable 레지스터

1.3. 레지스터의 설정 값 정의

■ **GPACON**

[입/출력/부가 기능 사용 허가 설정 레지스터]

이 레지스터는 GPIO pin 들을 입력 설정 할 것인지, 출력으로 설정 할 것인지, 부가적인 기능을 사용할 것인지를 결정한다.

0로 해당 bit가 설정되면 출력이 가능하게 되어 GPADAT 레지스트에 어떤 값을 넣어서 제어할 수 있다.

1로 해당 bit가 설정되면 부가적인 기능이 가능하게 된다.

이 레지스터는 RESET시에는 1로 bit가 설정된다.

이 레지스트는 NAND, nCS, address 신호의 부가적인 기능을 사용하므로 이 레지스트를 변경하여 출력으로 사용하지 않는 것이 바람직하다.

■ **GPBCON - GPHCON**

[입/출력/부가 기능 사용 허가 설정 레지스터]

이 레지스터는 GPIO pin 들을 입력 설정 할 것인지, 출력으로 설정 할 것인지, 부가적인 기능을 사용할 것인지를 결정한다.

0으로 해당 bit가 설정되면 입력 전용으로 설정되어 GPBDAT – GPHDAT 레지스트를 통해서 GPIO pin 상태를 읽을 수 있다.

1로 해당 bit가 설정되면 출력이 가능하게 되어 GPBDAT – GPHDAT 레지스트에 어떤 값을 넣어서 제어할 수 있다.

2로 해당 bit가 설정되면 부가적인 기능이 가능하게 된다.

이 레지스터는 RESET시에는 0으로 bit가 설정된다.

■ **GPBUP - GPHCON**

[Pull-up 레지스터의 enable/disable 설정]

이 레지스터는 GPIO의 Pull-up 을 허가 또는 금지를 설정한다.

1로 해당 bit가 설정되면 Pull-up 레지스터는 disable 된다.

0으로 bit를 설정하면 Pull-up 레지스터는 enable 된다.

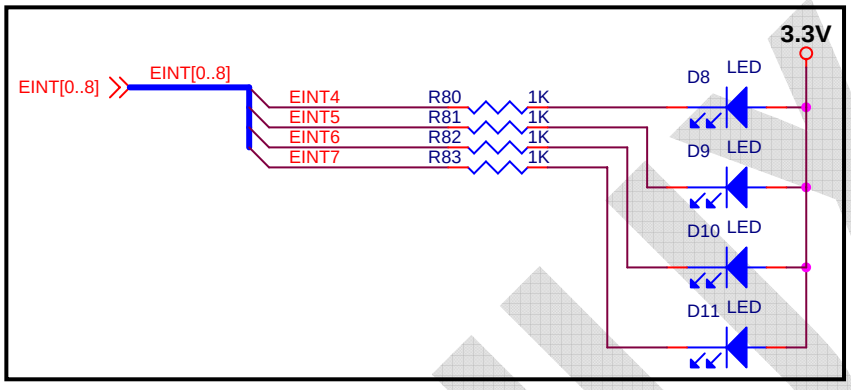
이 레지스터는 RESET시에는 0으로 bit가 설정된다.

위와 같은 GPIO 레지스터들은 커널의 **include/asm-arm/arch-s3c2410/regs-gpio.h**에 정의되어 있다.

1.4. GPIO를 이용한 LED 구동 및 스위치 동작 확인

간단하게 타겟보드에서 디버깅용으로 달려있는 4개의 LED를 가지고 GPIO의 입출력을 TEST 하는 방법을 소개하겠다.

■ **회로도**



회로도에서 보면 타겟보드의 디버깅용으로 사용한 LED가 연결되어 있는 GPIO는 GPF4, GPF5, GPF6, GPF7 이다.

■ **준비 조건**

❖ 모든 작업 디렉토리는 다음과 같다.

커널 작업 디렉토리 : /project/ez-s2410/kernel/linux
디바이스 드라이버 디렉토리 : /project/ ez-s2410/gpio
어플리케이션 디렉토리 : /project/ ez-s2410/gpio
헤더파일 디렉토리 : /project/ ez-s2410/gpio

❖ NFS(Network File System)를 사용할 경우에는 NFS가 마운트되어 있어야 한다.

❖ HowTo 문서 중 [커널 설치 및 패치]의 내용을 탐독하시고, 커널을 컴파일 해야 한다. 물론 제공한 CD에서 패치한 커널을 복사하여 사용하여도 상관없다.

문서 내용 중 " /project/ez-s2410/kernel/linux 이란 디렉토리에 설치하는 것을 가정하여 설명한다." 는 말이 있다. 이것은 Makefile을 만들 때 중요한 위치가 된다. 이 내용에 관해서는 아래 Makefile 만들기에서 설명하기로 한다.

다음 화면은 커널 컴파일 후 /project/ez-s2410/kernel/linux 의 리스트이다.
[.config 파일이 있는지 꼭 확인하기 바란다.]

```
root@arm6:/project/ez-s2410/kernel/linux
-rw-r--r--   1 root     root     18830  5월 12 10:38 .config
-rw-r--r--   1 root     root     18570  5월  6 23:23 .config.old
-rw-r--r--   1 root     root         4  6월 15 20:08 .version
-rwxrwxrwx   1 root     root     18691  7월 18  2004 COPYING
-rwxrwxrwx   1 root     root     86490  7월 18  2004 CREDITS
drwxrwxrwx  43 root     root      4096  5월 24 01:38 Documentation
-rwxrwxrwx   1 root     root     52664  7월 18  2004 MAINTAINERS
-rwxrwxrwx   1 root     root     37411  4월  8 00:01 Makefile
-rwxrwxrwx   1 root     root     13976  7월 18  2004 README
-rwxrwxrwx   1 root     root      2815  7월 18  2004 REPORTING-BUGS
drwxrwxrwx  23 root     root      4096  7월 18  2004 arch
drwxrwxrwx   2 root     root      4096  6월 20 18:21 crypto
drwxrwxrwx  45 root     root      4096  6월 20 18:21 drivers
drwxrwxrwx  53 root     root      4096  6월 20 18:21 fs
drwxrwxrwx  36 root     root      4096  9월 21  2004 include
drwxrwxrwx   2 root     root      4096  6월 20 18:21 init
drwxrwxrwx   2 root     root      4096  6월 20 18:21 ipc
drwxrwxrwx   3 root     root      4096  6월 20 18:21 kernel
drwxrwxrwx   4 root     root      4096  6월 20 18:21 lib
drwxrwxrwx   2 root     root      4096  6월 20 18:21 mm
drwxrwxrwx  32 root     root      4096  6월 20 18:21 net
drwxrwxrwx   8 root     root      4096  6월 20 18:21 scripts
drwxrwxrwx   3 root     root      4096  6월 20 18:21 security
drwxrwxrwx  15 root     root      4096  6월 20 18:21 sound
drwxrwxrwx   2 root     root      4096  6월 20 18:21 usr
[root@arm6 linux]#
[영어][완성][두벌식]
```

❖ S3C2410를 위한 ARM 용 크로스 컴파일러가 설치되어 있어야 한다.
컴파일에 필요한 파일들을 확인하자.

```
root@arm6:/usr/bin
[root@arm6 bin]# ls -al arm-linux-*
-rwxr-xr-x  1 root root 1854067  3월 30 17:18 arm-linux-addr2line
-rwxr-xr-x  1 root root 1911297  3월 30 17:18 arm-linux-ar
-rwxr-xr-x  1 root root 3301419  3월 30 17:18 arm-linux-as
-rwxr-xr-x  2 root root  279652  3월 30 17:59 arm-linux-c++
-rwxr-xr-x  1 root root 1806592  3월 30 17:18 arm-linux-c++filt
-rwxr-xr-x  1 root root  279467  3월 30 17:59 arm-linux-cpp
-rwxr-xr-x  2 root root  279652  3월 30 17:59 arm-linux-g++
-rwxr-xr-x  2 root root  279084  3월 30 17:59 arm-linux-gcc
-rwxr-xr-x  2 root root  279084  3월 30 17:59 arm-linux-gcc-3.4.3
-rwxr-xr-x  1 root root   15804  3월 30 17:59 arm-linux-gccbug
-rwxr-xr-x  1 root root   98796  3월 30 17:59 arm-linux-gcov
-rwxr-xr-x  1 root root 7800760  4월  9 19:11 arm-linux-gdb
-rwxr-xr-x  1 root root 2628605  3월 30 17:18 arm-linux-ld
-rwxr-xr-x  1 root root 1889956  3월 30 17:18 arm-linux-nm
-rwxr-xr-x  1 root root 2396280  3월 30 17:18 arm-linux-objcopy
-rwxr-xr-x  1 root root 2534819  3월 30 17:18 arm-linux-objdump
-rwxr-xr-x  1 root root 1911296  3월 30 17:18 arm-linux-ranlib
-rwxr-xr-x  1 root root  386348  3월 30 17:18 arm-linux-readelf
-rwxr-xr-x  1 root root 1681131  4월  9 19:11 arm-linux-run
-rwxr-xr-x  1 root root 1758830  3월 30 17:18 arm-linux-size
-rwxr-xr-x  1 root root 1734300  3월 30 17:18 arm-linux-strings
-rwxr-xr-x  1 root root 2396279  3월 30 17:18 arm-linux-strip
[영어][완성][두벌식]
```

■ 디바이스 드라이버

❖ *디바이스 드라이버용 Makefile을 만든다.*

작업 공간은 /project/ ez-s2410/gpio 디렉토리이다.

 # vi Makefile

```
#
# kernel 2.6 driver Makefile
#

obj-m    := dev_gpio.o

KDIR    := /project/ez-s2410/kernel/linux
PWD       := $(shell pwd)

SP_APP    = app_gpio

default:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
	cp -f $(obj-m:.o=.ko) /nfs/sap/.

	arm-linux-gcc -o $(SP_APP) $(SP_APP).c
	cp -f $(SP_APP) /nfs/sap

clean :
	rm -rf *.ko
	rm -rf *.mod.*
	rm -rf .*.cmd
	rm -rf *.o
```

obj-m   := dev_gpio.o

kernel-2.6대에서 모듈 컴파일을 위해서 필요하다. 모듈은 **.ko** 형태로 생성된다.

KDIR = /project/ez-s2410/kernel/linux

컴파일에 사용될 커널 디렉토리를 적어 준다.

$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

모듈을 컴파일 한다.

arm-linux-gcc -o $(SP_APP) $(SP_APP).c

어플리케이션을 컴파일 한다.

❖ *디바이스 드라이버를 만든다. [dev_gpio.c]*

작업 공간은 /project/ ez-s2410/gpio 디렉토리이다.

 # vi gpio.c

```
/*--------------------------------------------------------------------------
 파 일 : dev_gpio.c
 설 명 : 디바이스 드라이버 예제
         kernel 2.6 버젼으로 작성되었다.

 작 성 :

      ez-s2410 보드의 GPIO핀 GPF[4..7] 에는 디버그용 LED 가 연결되어있다.

      GPF4, GFP5 : 입력
      GPF6, GFP7 : 출력

--------------------------------------------------------------------------*/
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/interrupt.h>
#include <linux/wait.h>
#include <linux/ioport.h>
#include <linux/slab.h>
#include <linux/poll.h>
#include <linux/proc_fs.h>
#include <asm/system.h>
#include <asm/uaccess.h>
#include <asm/hardware.h>
#include <asm/irq.h>
#include <asm/ioctl.h>
#include <asm/unistd.h>
#include <asm/io.h>

#include <asm/arch/regs-gpio.h>

#include "dev_gpio.h"


/* 로컬상수 정의 ------------------------------------------------------------*/
#define    DEV_NAME          "dev-gpio"
#define    DEV_VERSION       DEV_NAME" V01"

/* 로컬변수 정의 ------------------------------------------------------------*/
static int showmsg = 0;
static int major     = DEV_MAJOR_DEF;


/* 매크로 정의 --------------------------------------------------------------*/

#define    reg_s2410(x)                  *(volatile unsigned long *)(x)
#define    set_GPIO_IN(reg, nr)          reg &= ~(3<<(nr)*2)
#define    set_GPIO_OUT(reg, nr)         reg &= ~(3<<(nr)*2); reg |= (1<<(nr)*2)

#define    GPIO_IN(reg,nr)               (reg & (1<<(nr)) ? 1:0)
#define    GPIO_OUT(reg,nr,val)          reg &= ~(1<<(nr)); reg |= (val&0x1)<<(nr);
```

```
/*implementation  ======================================================*/

/*--------------------------------------------------------------------------
  설 명 : GPIO 입출력을 설정한다.
  주 의 :
--------------------------------------------------------------------------*/
void        hw_gpf_init( void )
{
            set_GPIO_IN(    reg_s2410(S3C2410_GPFCON), 4 );
            set_GPIO_IN(    reg_s2410(S3C2410_GPFCON), 5 );

            set_GPIO_OUT( reg_s2410(S3C2410_GPFCON), 6 );
            set_GPIO_OUT( reg_s2410(S3C2410_GPFCON), 7 );

            // 내부 pull-up disable   7..4
            reg_s2410(S3C2410_GPFUP) |= 0xf0;

            // led off
            GPIO_OUT( reg_s2410(S3C2410_GPFDAT), 6, 1 );
            GPIO_OUT( reg_s2410(S3C2410_GPFDAT), 7, 1 );
}

/*--------------------------------------------------------------------------
  설 명 : GPIO 출력
  주 의 :
--------------------------------------------------------------------------*/
void        hw_gpf_out( int gpnr, int val )
{
            GPIO_OUT( reg_s2410(S3C2410_GPFDAT), gpnr, val );
}

/*--------------------------------------------------------------------------
  설 명 : GPIO 입력
  주 의 :
--------------------------------------------------------------------------*/
int         hw_gpf_in( int gpnr )
{
            return GPIO_IN( reg_s2410(S3C2410_GPFDAT), gpnr );
}

/*--------------------------------------------------------------------------
  설 명 : 어플리케이션에서 디바이스를  open 하였을때 호출되는 함수
  주 의 :
--------------------------------------------------------------------------*/
int         gpio_open( struct inode *inode, struct file *filp )
{
           try_module_get(THIS_MODULE);

           if (showmsg) printk(" %s OPEN\n", DEV_NAME );
           return 0;
}

/*--------------------------------------------------------------------------
  설 명 : 어플리케이션에서 디바이스를  close를 하였을때 호출되는 함수
  주 의 :
--------------------------------------------------------------------------*/
int         gpio_release( struct inode *inode, struct file *filp)
{
           module_put(THIS_MODULE);

           if (showmsg) printk(" %s CLOSE\n", DEV_NAME );
           return 0;
}
```

```
/*-------------------------------------------------------------------------
  설 명 : ioctl 함수
-------------------------------------------------------------------------*/
int           gpio_ioctl( struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg )
{
            switch (cmd)
            {
            case IOCTL_GPIO_IN    :
             if ( ( 4 <= arg ) && ( arg <= 5 ) )
             {
                        return    hw_gpf_in( arg );
             }
             return -1;

            case IOCTL_GPIO_OUT_SET :
             if ( ( 6 <= arg ) && ( arg <= 7 ) )
             {
                        hw_gpf_out( arg, 1 );
                        return 0;
             }
             return -1;

             case IOCTL_GPIO_OUT_CLR :
             if ( ( 6 <= arg ) && ( arg <= 7 ) )
             {
                        hw_gpf_out( arg, 0 );
                        return 0;
             }
             return -1;
            }

            return -EINVAL;
}
/*-------------------------------------------------------------------------
  드라이버에 사용되는 접근 함수에 대한 함수 포인터 구조체를 정의 한다.
  이 구조체는 fs.h에 정의 되어 있다.
-------------------------------------------------------------------------*/
static struct file_operations gpio_fops =
{
            .open       = gpio_open,
            .release = gpio_release,
            .ioctl      = gpio_ioctl,
};
/*-------------------------------------------------------------------------
  설 명 : init
-------------------------------------------------------------------------*/
int           gpio_init( void )
{
            // 장치를 등록한다. ==========================================
            major &= 0xff;
            if( !register_chrdev( major, DEV_NAME, &gpio_fops ) )
            {
                        printk(" register device %s OK (major=%d)\n", DEV_VERSION, major );
            }
            else
            {
                        printk(" unable to get major %d for %s \n", major, DEV_NAME );
                        return -EBUSY;
            }

             hw_gpf_init();

            return 0;
}
```

```
/*-------------------------------------------------------------------------
   설 명 : exit
   주 의 :
-------------------------------------------------------------------------*/
void         gpio_exit( void )
{
            unregister_chrdev( major, DEV_NAME );
            printk(" unregister %s OK\n", DEV_NAME );
}

/*-------------------------------------------------------------------------*/

module_init(gpio_init);
module_exit(gpio_exit);

MODULE_PARM(major    , "i");
MODULE_PARM(showmsg, "i");
MODULE_AUTHOR("freefrug@falinux.com");
MODULE_LICENSE("Dual BSD/GPL");

/* end */
```

❖ *헤더파일을 만든다. [gpio.h]*

작업 공간은 /project/ez-x5/test/gpio/include 디렉토리이다.

 # vi gpio.h

```
//-------------------------------------------------------------------------
// 파 일 : dev_gpio.c
// 설 명 : 디바이스 드라이버 예제
//             kernel 2.6.x 버젼으로 작성되었다.
//
// 작 성 : 오재경 freefrug@falinux.com
//-------------------------------------------------------------------------
#ifndef _DEV_GPIO_H_
#define _DEV_GPIO_H_

/* 모듈관련 ---------------------------------------------------------------*/
#define             DEV_MAJOR_DEF                   240

/* ioctl 함수의 command ---------------------------------------------------*/
#define             MAGIC_GPIO                      's'

#define             IOCTL_GPIO_IN                   _IOW( MAGIC_GPIO, 10, int )
#define             IOCTL_GPIO_OUT_SET              _IOW( MAGIC_GPIO, 11, int )
#define             IOCTL_GPIO_OUT_CLR              _IOW( MAGIC_GPIO, 12, int )


#endif   // _DEV_GPIO_H_
```

■ 어플리케이션

❖ 어플리케이션 프로그램을 만든다. [test.c]
작업 공간은 /project/ ez-s2410/gpio 디렉토리이다.

 # vi app_gpio.c

```
//-------------------------------------------------------------------------
// 파 일 : app_gpio.c
// 설 명 :
//-------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/signal.h>
#include <sys/ioctl.h>
#include <sys/poll.h>

#include "dev_gpio.h"

#define                 NODE_FILE                       "/dev/gpio"

//-------------------------------------------------------------------------
// 설명 : main
// 주의 :
//-------------------------------------------------------------------------
int         main( int argc, char **argv )
{
            int             fd;                     //   파일핸들

              fd = open( NODE_FILE, O_RDWR | O_NONBLOCK );
              if ( fd < 0 )
              {
                        perror( NODE_FILE" open :");
                        exit(0);
              }
              printf( "app_gpio start ...\n" );

              // led ON
              ioctl( fd, IOCTL_GPIO_OUT_CLR, 6 );     sleep(1);
              ioctl( fd, IOCTL_GPIO_OUT_CLR, 7 );     sleep(1);
              // led OFF
              ioctl( fd, IOCTL_GPIO_OUT_SET, 6 );
              ioctl( fd, IOCTL_GPIO_OUT_SET, 7 );

              while(1)
              {
                        if ( 0 == ioctl( fd, IOCTL_GPIO_IN, 4 ) )
                        {
                                  ioctl( fd, IOCTL_GPIO_OUT_CLR, 6 );     sleep(1);
                                  ioctl( fd, IOCTL_GPIO_OUT_SET, 6 );
                                  ioctl( fd, IOCTL_GPIO_OUT_CLR, 7 );     sleep(1);
                                  ioctl( fd, IOCTL_GPIO_OUT_SET, 7 );
                        }
```

```
                        if ( 0 == ioctl( fd, IOCTL_GPIO_IN, 5 ) )
                        {
                                ioctl( fd, IOCTL_GPIO_OUT_CLR, 6 );
                                ioctl( fd, IOCTL_GPIO_OUT_CLR, 7 );
                                break;
                        }
                }

                close( fd );

                printf( "app_gpio end\n" );
                return 0;
}
```

여기까지 EZ-S2410에서 GPIO를 이용한 TEST 할 수 있는 디바이스 드라이버와 어플리케이션을 만들었다. 이 소스는 제공한 CD에 들어있다.

■ **컴파일**

작업 공간은 /project/ez-x5/test/gpio/include 디렉토리이다.

 # make clean

 # make

위와 같이 컴파일이 정상적으로 수행이 되고 나면 다음과 같은 파일들이 생성된다. 여기서 필요한 모듈과 실행 파일은 다음과 같다.

모듈 : dev_gpio.ko
실행파일 : app_gpio

■ **타겟보드에 디바이스 드라이버 및 어플리케이션 다운로드하기**

타겟보드에 다운로드하는 방법은 두 가지가 있다.
첫번째는 NFS를 이용하는 방법이 있다. 이것은 디버깅을 하기에 유용하다.
두번째는 램디스크 이미지에 넣는 방법이 있다. 이것은 디버깅 작업을 할 때는 매우 귀찮은 작업이다.

첫번째 방법은 NFS를 마운트되어 있다는 전제조건 하에 마운트된 디렉토리에 컴파일한 dev_gpio.ko와 app_gpio파일을 복사하여 모듈을 적재하고 어플리케이션을 실행시키면 된다.

$ insmod dev_gpio.ko
$ ./ app_gpio

```
[root@falinux ~]$ mount -t nfs -o nolock 192.168.10.26:/nfs /mnt/nfs/
nfs warning: mount version older than kernel
[root@falinux ~]$
[root@falinux ~]$ cd /mnt/nfs/sap/
[root@falinux sap]$
[root@falinux sap]$ insmod dev_gpio.ko
Using dev_gpio.ko
  register device dev-gpio V01 OK (major=240)
[root@falinux sap]$
[root@falinux sap]$ ./app_gpio
/dev/gpio open :: No such file or directory
[root@falinux sap]$
[root@falinux sap]$ mknod /dev/gpio c 240 0
[root@falinux sap]$
[root@falinux sap]$ ./app_gpio
app_gpio start ...

[root@falinux sap]$
```

두번째 방법은 램디스크 이미지를 이용하는 방법에 대하여 설명하겠다.

❖ *램디스크 이미지 복사하기*

타겟보드에 올라가 있는 램디스크를 가져오자. 만일 없다면 제공한 CD에서 복사를 하자.

자세한 설명은 [10 장 램 디스크 이미지 제작]을 참고하기 바란다.
단지 여기서는 작업 내용 만을 적어놓겠다.

모든 작업 디렉토리는 /project/ez-s2410/ramdisk 이며, 디렉토리를 만든다.
 # mkdir /project/ez-s2410/ramdisk

램디스크 이미지를 복사한다.
 # cp -a /mnt/cdrom/sw/ramdisk/ramdisk-12M.gz /project/ez-s2410/ramdisk/

작업 디렉토리로 이동한다.
 # cd /project/ez-s2410/ramdisk

❖ *램디스크 이미지 압축 풀기*

램디스크 이미지 압축을 풀어 마운트 할 작업 디렉토리를 만든다.
 # mkdir ramdisk_dir

램디스크 이미지 압축을 푼다.
 # gzip –d ramdisk-12M.gz

압축을 푼 램디스크 이미지를 마운트한다.
 # mount –t ext2 –o loop ramdisk-12M ramdisk_dir

❖ *램디스크 이미지를 수정한다.*

디바이스 노드를 만든다.
 # cd ramdisk_dir
 # mknod /dev/gpio c 240 0

디바이스 드라이버와 어플리케이션 프로그램을 복사할 디렉토리를 만든다.

```
# pwd
   /project/ez-s2410/ramdisk/ramdisk_dir
# mkdir gpio
# cd gpio
# cp –a /project/ez-s2410/gpio/dev_gpio.ko   .
# cp –a /project/ez-s2410/gpio/app_gpio   .
```

❖ *램디스크 이미지 압축 하기*

원하는 대로 램디스크를 만든 후에 다음의 작업을 하여 압축한다.
마운트를 해제한다.

```
# pwd
  /project/ez-s2410/
# umount ramdisk_dir
```

램디스크 이미지 압축한다.

```
# gzip ramdisk-12M
```

❖ *램디스크 이미지를 타겟보드로 다운로드 한다.*

다운로드 하는 방법은 앞 장에 설명되어 있다. 이를 참고하기 바란다.

■ **모듈 적재 및 프로그램 실행**

타겟보드에 램디스크를 다운로드 하였다면 타겟보드를 다시 부팅하고 로그인하여 타겟보드에 gpio 디렉토리가 생성되고, 이 디렉토리 안에 **dev_gpio.ko** 와 **app_gpio** 파일이 존재할 것이다.

❖ *모듈 적재하기*

모듈 적재는 insmod 명령을 사용한다.

```
$ cd /gpio/
$ insmod dev_gpio.ko
```

만약 /dev/gpio라는 파일 시스템이 없으면 만들어 주어야 한다.

```
$ ls /dev/   | grep gpio
$ mknod   /dev/gpio   c   240   0
```

모듈 해지는 rmmod 명령을 사용한다.

$ rmmod dev_gpio.ko

다음은 위의 내용을 실행한 화면이다.

```
[root@falinux ~]$ mount -t nfs -o nolock 192.168.10.26:/nfs /mnt/nfs/
nfs warning: mount version older than kernel
[root@falinux ~]$
[root@falinux ~]$ cd /mnt/nfs/sap/
[root@falinux sap]$
[root@falinux sap]$ insmod dev_gpio.ko
Using dev_gpio.ko
  register device dev-gpio V01 OK (major=240)
[root@falinux sap]$
[root@falinux sap]$ ./app_gpio
/dev/gpio open :: No such file or directory
[root@falinux sap]$
[root@falinux sap]$ mknod /dev/gpio c 240 0
[root@falinux sap]$
  [root@falinux sap]$ rmmod dev_gpio
  unregister dev-gpio OK
[root@falinux sap]$
```

❖ *어플리케이션 프로그램 실행하기*

모듈이 적재되었고, 디바이스 드라이버 노드가 만들어져 있다면 어플리케이션 프로그램을 실행시켜 GPIO를 이용한 LED의 입/출력 TEST를 하자.

$ insmod gpio_dev.o

$ ./app_gpio

```
[root@falinux ~]$ mount -t nfs -o nolock 192.168.10.26:/nfs /mnt/nfs/
nfs warning: mount version older than kernel
[root@falinux ~]$
[root@falinux ~]$ cd /mnt/nfs/sap/
[root@falinux sap]$
[root@falinux sap]$ insmod dev_gpio.ko
Using dev_gpio.ko
  register device dev-gpio V01 OK (major=240)
[root@falinux sap]$
[root@falinux sap]$ ./app_gpio
app_gpio start ...

[root@falinux sap]$
```

❖ *참고사항*

위에서 모두 설명한 내용이지만 다시 한번 참고하기 바란다.

1) insmod dev_gpio.ko 실행 시 다음과 같은 경고 메시지

```
[root@falinux sap]$ insmod dev_gpio.ko
Using dev_gpio.ko
Warning: loading gpio_dev will taint the kernel: no license

  See http://www.tux.org/lkml/#export-tainted for information
about tainted modules
```

이 경고메시지는 디바이스 드라이버의 modules license에 관련한 경고이다.

이 문제의 해결은 디바이스 드라이버의 마지막에 다음의 코드를 추가해 주면 된다. [FALINUX에서는 이 부분이 제거된 상태로 제공된다]

MODULE_LICENSE("Dual BSD/GPL ");

```
MODULE_PARM(major    , "i");
MODULE_PARM(showmsg, "i");
MODULE_AUTHOR("freefrug@falinux.com");
MODULE_LICENSE("Dual BSD/GPL");
```

2) 어플리케이션 실행 시 GPIO Open Fail이 나올 경우

```
[root@falinux sap]$ ./app_gpio
/dev/gpio open :: No such file or directory
```

이 문제는 디바이스 장치 파일이 없어서 나는 에러 메시지이다.
다음과 같이 파일 시스템을 /dev 디렉토리내에 만들어 주면 된다.

[root@falinux sap]$ mknod /dev/gpio c 240 0