Device Driver

모듈 프로그래밍

- 모듈
  - 하나의 오브젝트 파일(*.o, *.ko)
  - 실행중인 커널에 동적으로 적재되거나 제거
  - 이벤트 처리(Event handling) 형태의 프로그램 방식
    - Main() 함수가 없다
    - Startup, cleanup 함수 존재

- 모듈 프로그램의 이점
  - 효과적인 메모리 사용
  - 커널 전체를 다시 컴파일하지 않고 커널의 일부분 또는 디바이스 드라이버를 교체할 수 있음
- 모듈 프로그램의 단점
  - namespace pollution
    - kernel 에 삽입되어 확장된 커널이 된다.
    - 모듈을 개발하다 보면 global 변수 혹은 function 명과 쉽게 구별되지 않는다.
    - 기존 global 변수 혹은 function 명이 충돌되기 쉽다.

모듈 프로그래밍의 예-1

- 간단한 모듈프로그램 예 (hello.c)

```
/* Module example hello.c */

#include <linux/kernel.h>          /* 커널에서 수행될 때 필요한 헤더 파일 */
#include <linux/module.h>          /* 모듈에 필요한 헤더 파일 */
#include <linux/init.h>            /* module_init(), module_exit() 매크로 정의 */

static int module_begin(void)      /* 모듈이 설치될 때 초기화 수행,
                                    insmod 할 때 수행되는 함수 */
{
  printk("<1> Hello, Embedded  System\n");
  return 0;
}

static void module_end(void)    /*   모듈이 제거될 때 반환 작업 수행,
                                    rmmod 할 때 수행되는 함수*/
{
 printk("<1> Goodbye ^^ \n");
}

module_init(module_begin);
module_exit(module_end);
```

모듈 프로그래밍의 예-2

- 간단한 모듈프로그램 예 (HOST 2.4용 Makefile)

```
#Makefile for a basic kernel module

COMPILE :=gcc

#Modify this statement to your kernel directory
INCLUDEDIR := /usr/src/linux-2.4/include/

MODCFLAGS := -Wall -O2 -DMODULE -D__KERNEL__ -DLINUX -I$(INCLUDEDIR)

hello-x86.o : hello.c
        $(COMPILE)  $(MODCFLAGS)  -c  hello.c  -o  hello-x86.o

clean:
        rm  -rf  *.o
```

```
# make                          (모듈 컴파일)
# insmod  hello-x86.o           (모듈을 커널에 적재)
# more  /proc/modules           (적재된 모듈 보기)
# rmmod  hello-x86  (모듈 삭제하기)
```

모듈 프로그래밍의 예-2

- 간단한 모듈프로그램 예 (TARGET 2.6용 Makefile)

```
#Makefile for a basic kernel module

obj-m   := hello.o

KDIR    := /root/acumen/kernel/linux-2.6.9-acumen-v1.4
PWD     := $(shell pwd)

default:
	$(MAKE)  -C  $(KDIR)  SUBDIRS=$(PWD)  modules
clean:
	rm  -rf  *.ko
	rm  -rf  *.mod.*
	rm  -rf  *.o
```

```
# make                    (모듈 컴파일)
# insmod  hello.ko        (모듈을 커널에 적재)
# more  /proc/modules     (적재된 모듈 보기)
# rmmod  hello            (모듈 삭제하기)
```

모듈 관련 리눅스 명령어

- 모듈 명령어 요약

명령어	용 도
insmod	module을 설치(install)
rmmod	실행중인 modules을 제거(unload)
lsmod	Load된 module들의 정보를 표시
depmod	커널 내부에 적재된 모듈간의 의존성을 검사한다.
modprobe	모듈간 의존성을 검사하여 그 결과 누락된 다른 모듈을 찾아서 적재한다.
modinfo	목적화일을 검사해서 관련된 정보를 표시

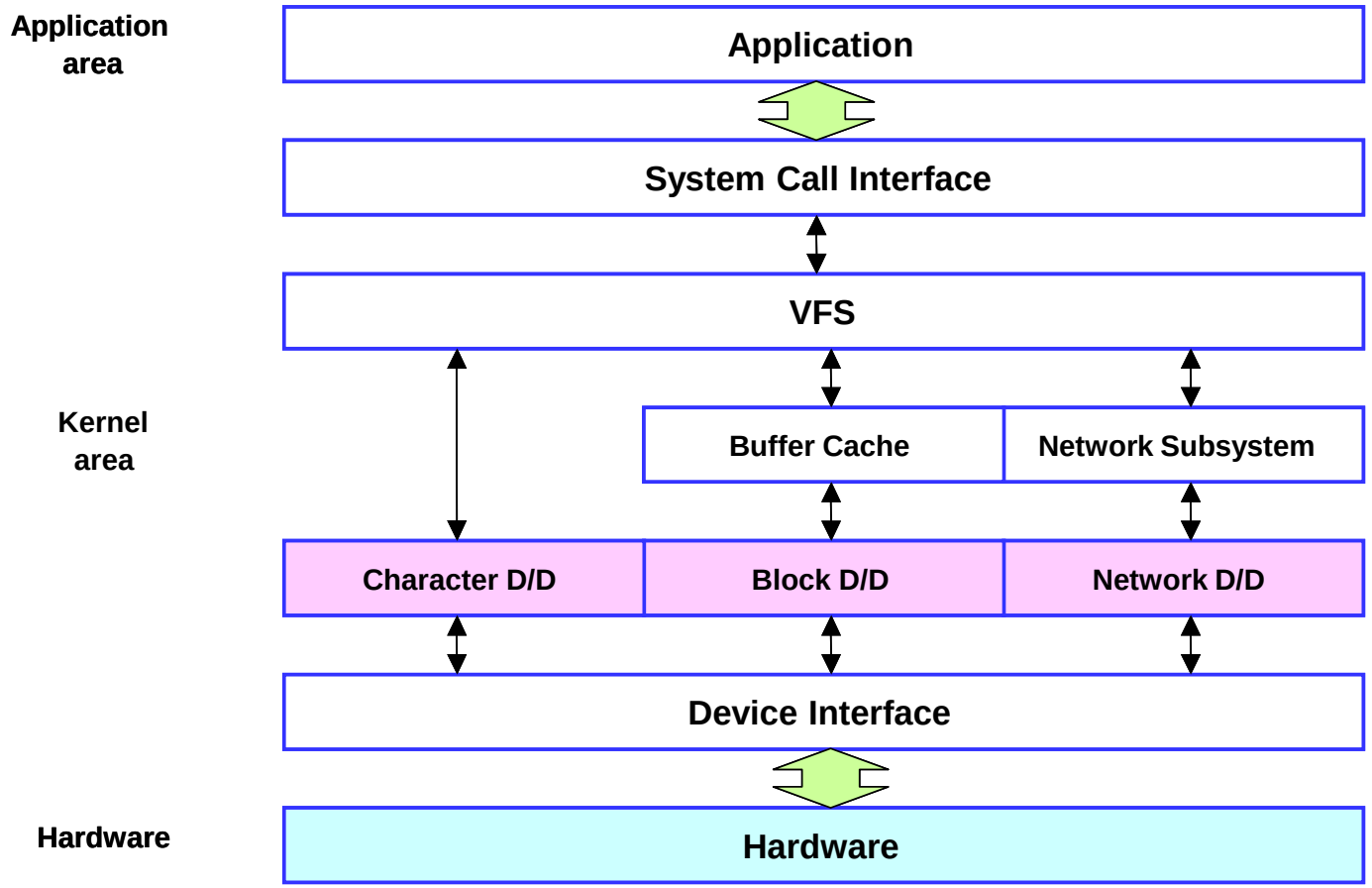
디바이스 드라이버

- 정의
  - 디바이스와 시스템 메모리 간에 데이터의 전달을 담당하는 커널의 내부 기능
- 구조
  - 파일 시스템과 인터페이스
  - 디바이스와 인터페이스
  - 드라이버 초기화 인터페이스

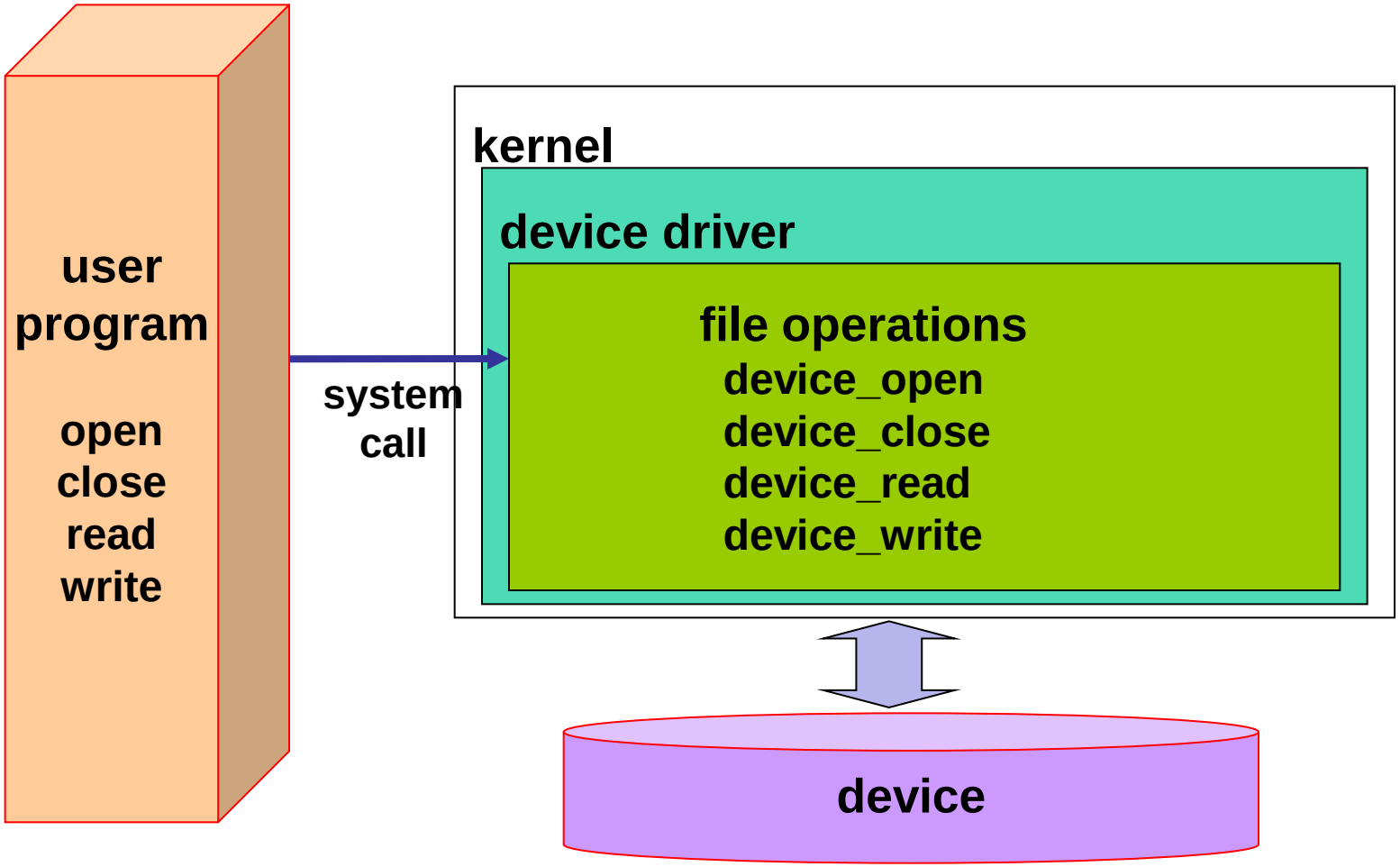
디바이스 드라이버 종류

- 문자 디바이스 드라이버
  - 파일 시스템에서 노드 형태로 존재
  - 파일처럼 취급하고 접근하여 직접 읽기/쓰기를 수행
  - 대부분 순차적인 접근 가능, 임의의 크기로 데이터 전송
  - 콘솔, 키보드, 시리얼 포트 드라이버 등
- 블록 디바이스 드라이버
  - 파일 시스템에서 노드 형태로 존재
  - 버퍼 캐쉬 이용
  - 블록 단위의 데이터 이동
  - 플로피 디스크, 하드 디스크, CDROM 드라이버 등
- 네트워크 디바이스 드라이버
  - 파일 형태가 아닌 특별한 인터페이스 사용(대응하는 장치파일이 없다.)
  - 스트림 기반이 아닌 패킷 단위의 접근
  - 이더넷 디바이스 드라이버(eth0)

리눅스 구조(Architecture)-1



리눅스 구조(Architecture)-2

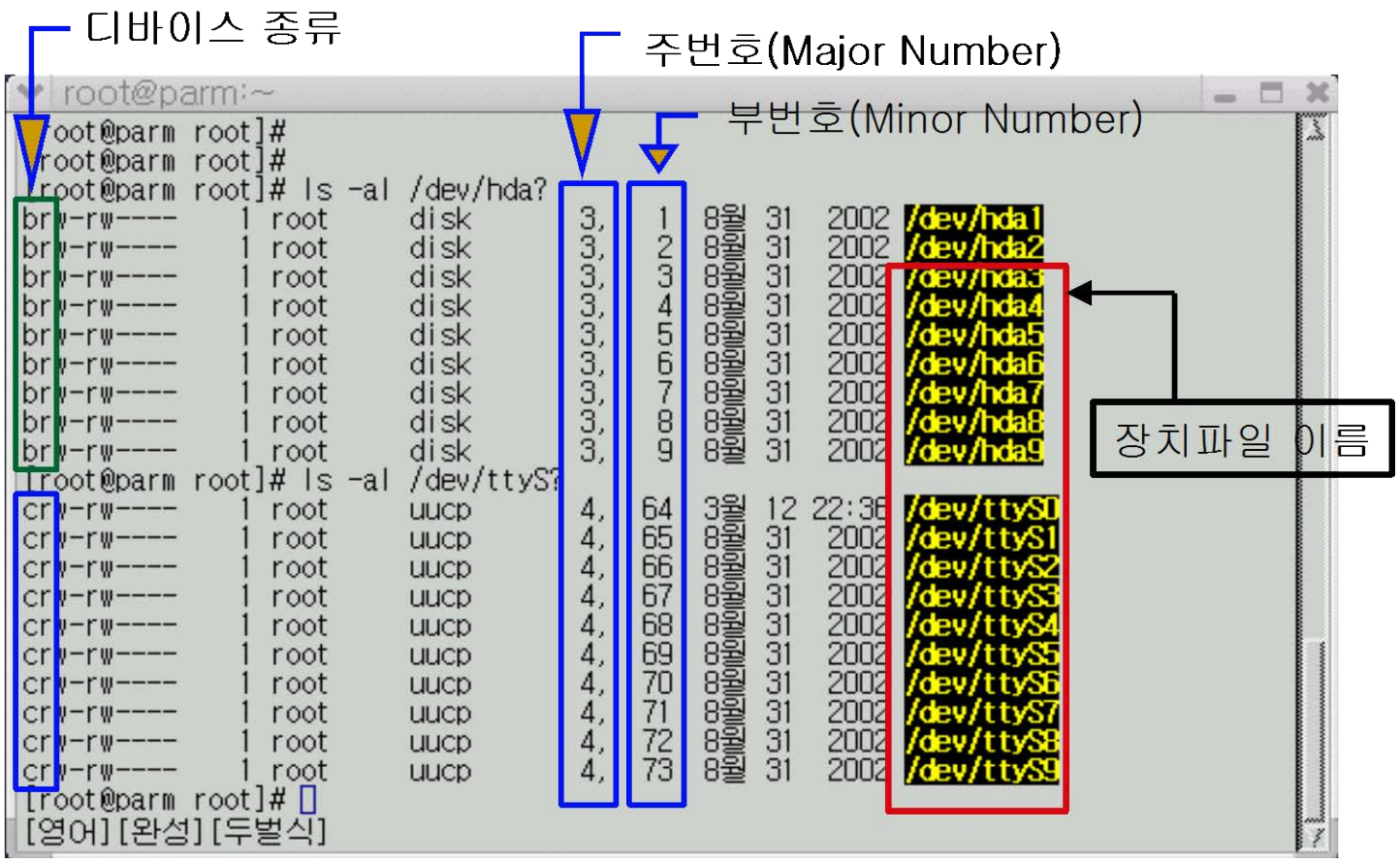


디바이스 파일(Device File)

- 디바이스 파일
  - 디바이스 드라이버를 접근하는 통로
  - 장치 파일의 inode는 장치 유형(type),주 번호(major number), 부번호(minor number)로 구성된다.
  - mknod 명령으로 디바이스 드라이버에 접근할 수 있는 장치 파일 생성
    - mknod [device file name] [type] [major] [minor]
    - mknod --mode=766 /dev/test c 240 0

- 주번호와 부번호
  - 주 번호 : 커널이 디바이스와 연관된 드라이버를 구분하는 데 사용
  - 부 번호 : 드라이버가 각 디바이스를 구분하는 데 이용
  - documentation/devices.txt

디바이스 드라이버 번호


디바이스 종류
주번호(Major Number)
부번호(Minor Number)
root@parm:~
root@parm root]#
root@parm root]#
root@parm root]# ls -al /dev/hda?
brw-rw---- 1 root disk 3, 1 8월 31 2002 /dev/hda1
brw-rw---- 1 root disk 3, 2 8월 31 2002 /dev/hda2
brw-rw---- 1 root disk 3, 3 8월 31 2002 /dev/hda3
brw-rw---- 1 root disk 3, 4 8월 31 2002 /dev/hda4
brw-rw---- 1 root disk 3, 5 8월 31 2002 /dev/hda5
brw-rw---- 1 root disk 3, 6 8월 31 2002 /dev/hda6
brw-rw---- 1 root disk 3, 7 8월 31 2002 /dev/hda7
brw-rw---- 1 root disk 3, 8 8월 31 2002 /dev/hda8
brw-rw---- 1 root disk 3, 9 8월 31 2002 /dev/hda9
[root@parm root]# ls -al /dev/ttyS?
crw-rw---- 1 root uucp 4, 64 3월 12 22:36 /dev/ttyS0
crw-rw---- 1 root uucp 4, 65 8월 31 2002 /dev/ttyS1
crw-rw---- 1 root uucp 4, 66 8월 31 2002 /dev/ttyS2
crw-rw---- 1 root uucp 4, 67 8월 31 2002 /dev/ttyS3
crw-rw---- 1 root uucp 4, 68 8월 31 2002 /dev/ttyS4
crw-rw---- 1 root uucp 4, 69 8월 31 2002 /dev/ttyS5
crw-rw---- 1 root uucp 4, 70 8월 31 2002 /dev/ttyS6
crw-rw---- 1 root uucp 4, 71 8월 31 2002 /dev/ttyS7
crw-rw---- 1 root uucp 4, 72 8월 31 2002 /dev/ttyS8
crw-rw---- 1 root uucp 4, 73 8월 31 2002 /dev/ttyS9
[root@parm root]#
[영어][완성][두벌식]
장치파일 이름

Linux Major Number List

Major	Character devices	Block devices
0	*unnamed for NFS, network and so on*	
1	Memory device(mem)	RAM disk
2		Floppy disks(fd*)
3		IDE hard disks(hd*)
4	Terminals	
5	Terminals & AUX	
6	Parallel Interfaces	
7	Virtual Consoles(vcs*)	
8		SCSI hard disks(sd*)
9	SCSI tapes(st*)	
10	Bus mice(bm, psaux)	
11		SCSI CD-ROM(scd*)
12	QIC02 tape	
13	PC Speaker driver	XT 8-bit hard disks(xd*)
14	Sound cards	BIOS hard disk support
15	Joystick	Cdu31a/33a CD-ROM
-		
19	Cyclades drivers	Double compressing driver
20	Cyclades drivers	
21	SCSI generic	
22		2nd IDE interface driver
23		Mitsumi CD-ROM(mcd*)
24		Sony 535 CD-ROM
25		Matsushita CD-ROM 1

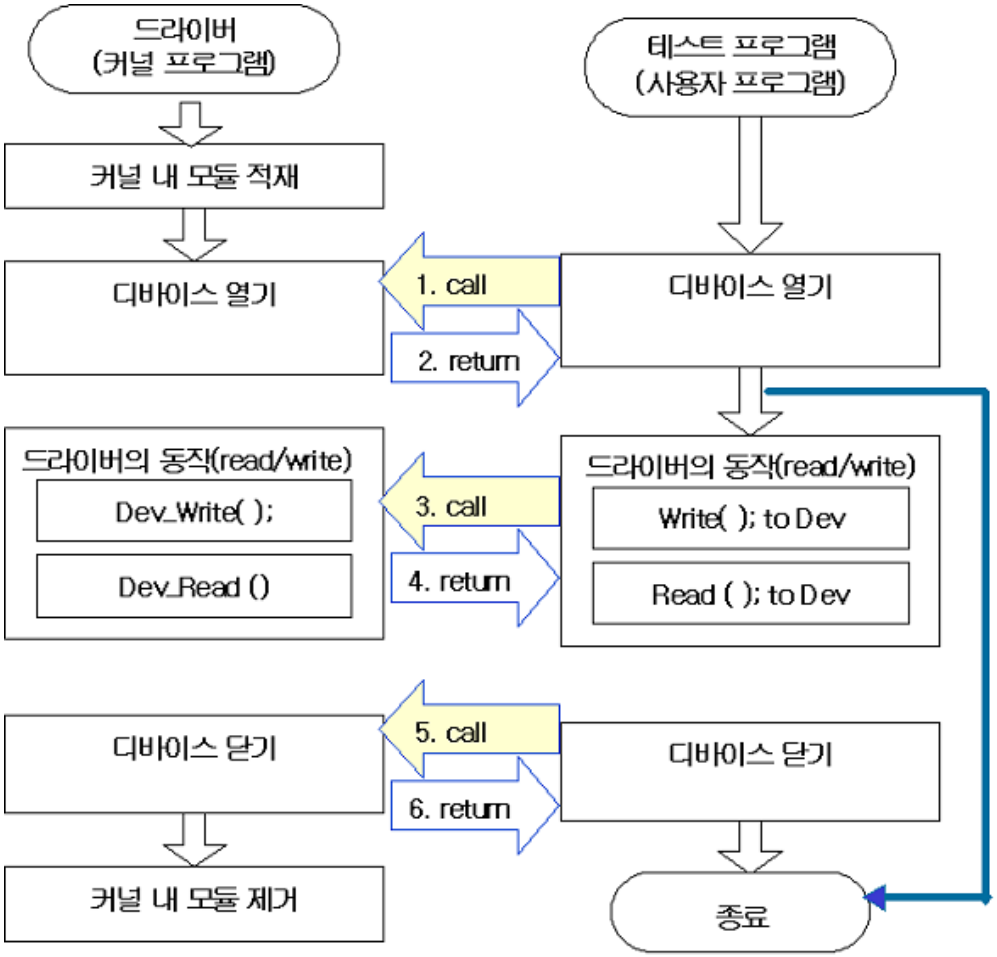
참조
$LinuxSrc/Documentation/devices.txt
$LinuxSrc/include/linux/major.h

Polling & Interrupt

- Polling
  - 요청한 작업이 끝났는지를 알기 위해 장치의 상태가 변할 때까지 장치의 상태 레지스터를 계속해서 주기적으로 읽는 것.
  - 디바이스 드라이버는 커널의 한 부분이기 때문에, 만약 드라이버가 폴링만 하려고 한다면 장치가 작업을 마칠 때까지 커널의 다른 부분이 수행될 수 없을 것이다.
  - 폴링을 하는 디바이스 드라이버는 시스템 타이머를 이용하여 어느 정도 시간이 지나면 커널이 디바이스 드라이버에 있는 한 루틴을 부르도록 한다. 이 타이머 루틴은 명령이 수행 되었는지 상태를 검사한다. 이는 리눅스의 플로피 드라이버에서 사용하는 방법이다.

- Interrupt
  - 타이머를 이용하는 폴링은 좋은 방법이지만, 이보다 더 효과적인 방법으로 인터럽트를 사용하는 것이 있다.
  - 제어하는 하드웨어 장치가 서비스를 받아야 할 때 하드웨어 인터럽트를 발생하는 것이 인터럽트를 이용한 디바이스 드라이버이다. 예를 들어, 이더넷 디바이스 드라이버는 네트웍에서 이더넷 패킷을 받을 때마다 인터럽트를 발생한다. 리눅스 커널은 이 인터럽트를 하드웨어 장치에서 올바른 디바이스 드라이버로 전달할 수 있어야 한다. 이는 디바이스 드라이버가 커널에 인터럽트를 사용하겠다고 등록함으로써 이루어진다. 드라이버는 인터럽트 처리 루틴의 주소와 자신이 사용하고 싶은 인터럽트 번호를 커널에 등록한다. 현재 디바이스 드라이버가 어떤 인터럽트를 사용하고 있으며, 그 인터럽트가 얼마나 많이 발생했는지 알려면, /proc/interrupts 파일을 보면 된다.

문자 디바이스 드라이버 동작



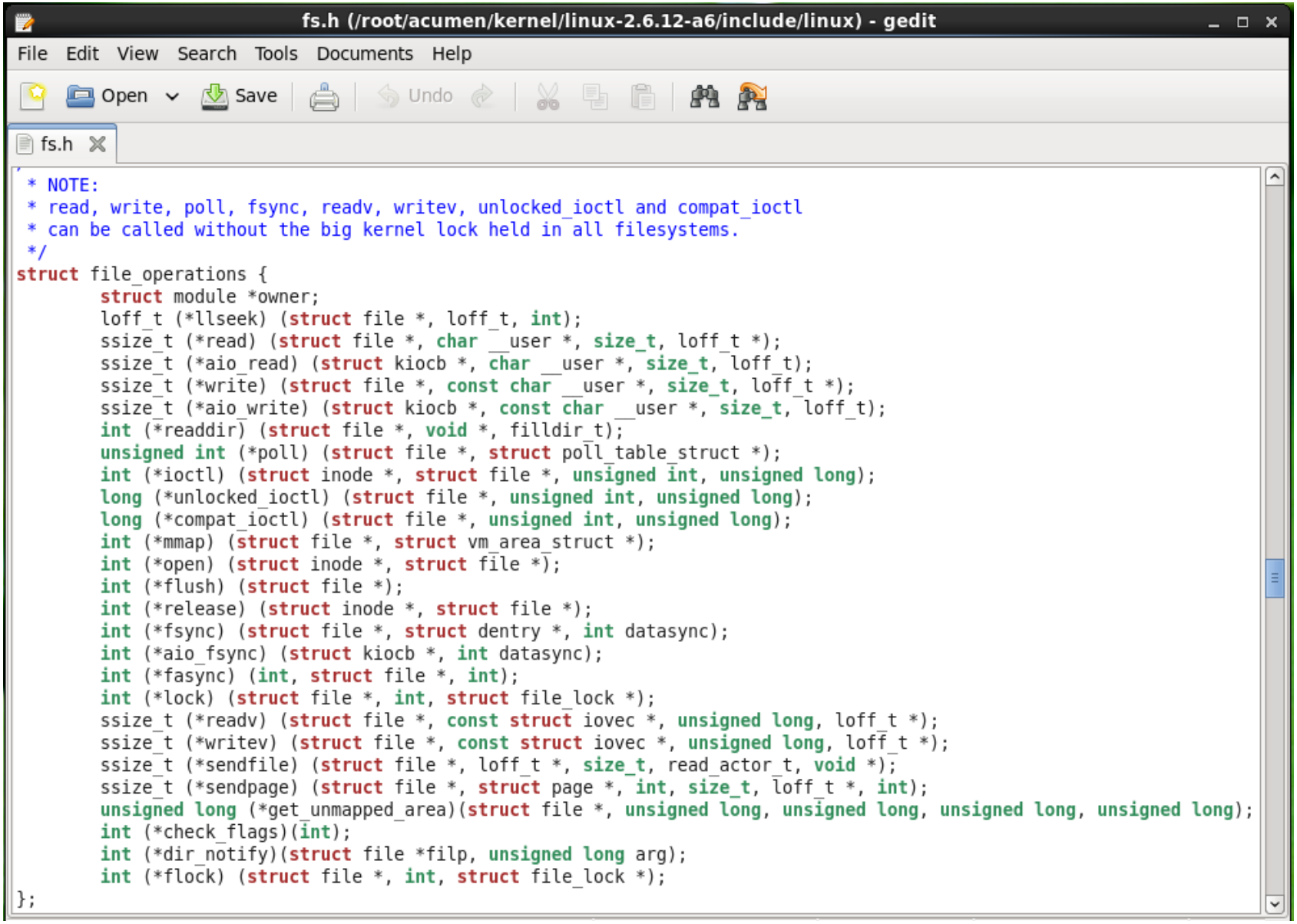
문자 디바이스 제작방법 - 1

- 문자 디바이스 드라이버 제작법
  - 외부와 디바이스 드라이버는 파일 인터페이스를 통해서 연결
  - 디바이스 드라이버는 file_operations를 제공함으로써 구현
  - 디바이스 드라이버는 자신을 구별하기 위해 고유의 major number를 사용
- 장치의 등록과 해제
  - 등록 : int register_chrdev(unsigned int major, const char *name, struct file_operations *fops)
    - major : 등록할 major number. 0이면 사용하지 않는 번호 중 동적으로 할당
    - name : 장치의 이름
    - fops : 장치에 대한 파일 연산 함수들
  - 해제 : int unregister_chrdev(unsigned int major, const char *name)

문자 디바이스 제작방법 - 2

- file_operations의 구조체 원형 -<linux/fs.h>참조

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void __user *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long,
            unsigned long, unsigned long);
};
```




```
 * NOTE:
 * read, write, poll, fsync, readv, writev, unlocked_ioctl and compat_ioctl
 * can be called without the big kernel lock held in all filesystems.
 */
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*aio_fsync) (struct kiocb *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
        int (*check_flags)(int);
        int (*dir_notify)(struct file *filp, unsigned long arg);
        int (*flock) (struct file *, int, struct file_lock *);
};
```

문자 디바이스 제작방법 - 3

- onwer:어떤 모듈로 올라간 디바이스 드라이버와 연관을 가지는지를 나타내는 포인터
- lseek : 파일에서 현재의 read/write 위치를 옮기며 새로운 위치가 양수 값으로 return된다. error는 음수가 return 된다.
- read : 디바이스에서 데이터를 가져오기 위해 사용된다.
- write : 디바이스에 데이터를 보낸다.
- readdir : 디바이스 노드에서는 NULL이어야 하며 디렉토리에 대해서 사용한다.
- poll : 현재 프로세스에 대기 큐를 넣는다 (2.0에서 select_wait과 같다)
- ioctl : 디바이스에 종속적인 명령을 만들 수 있도록 한다.(플로피의 한 트랙을 포맷하는 명령)
- mmap : 디바이스 메모리를 프로세스 메모리에 맵핑 시키도록 요청에 사용
- open : 디바이스 노드에 대해 수행되는 첫 번째 동작,application의해 장치가 열릴 때 호출
- flush : 열린 디바이스를 닫기 전에 모든 데이터를 쓰도록 하기 위해 사용
- release : 노드를 닫을 때 수행된다
- fsync : 디바이스에 대한 모든 연산의 결과를 지연하지 않고 즉시 일어나도록 한다.
- fasync : FASYNC 플레그에 변화가 있는 디바이스를 확인하기 위해 사용한다.
- check_media_change : 블록 디바이스에 사용되는데 플로피처럼 제거가능 미디어에 사용
- revalidate : check_media_change와 관련 버퍼 캐쉬의 관리에 상관이 있다.
- lock : 파일체계에서만 유효하며 사용자가 파일을 잠그게 하기 위해 사용 한다.

문자 디바이스 제작방법 - 4

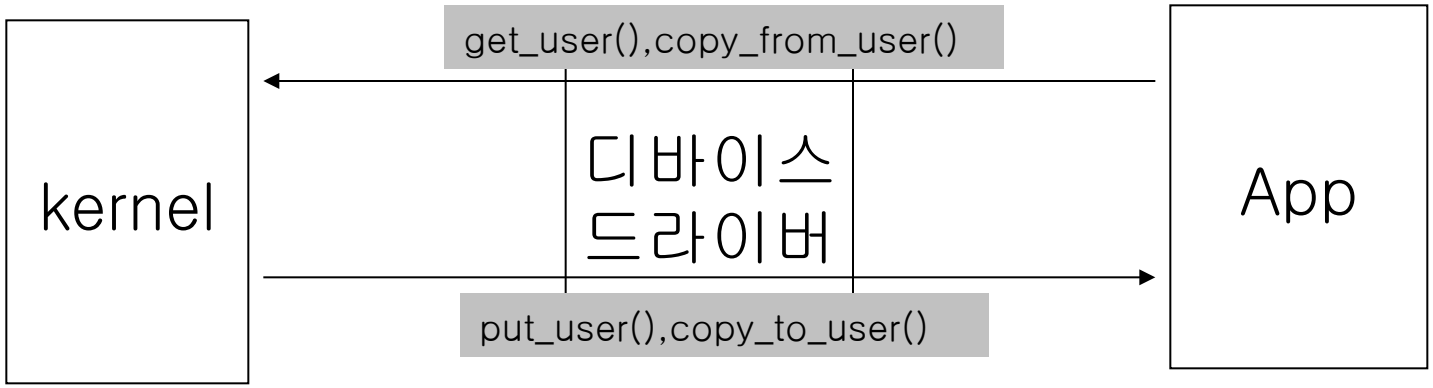
- Open에서 할 일
  - 처음으로 장치를 연 경우 장치 초기화
  - Minor번호를 확인하고 필요한 경우 f_op 포인터 수정
  - 필요한 경우 메모리를 할당 받아 filp-private_data삽입
  - 참조 횟수(usage count)를 증가
- Release에서 할 일
  - 참조 횟수를 감소
  - Filp-private_data에 할당된 데이터가 있으면 삭제
  - Close하는 경우 장치 종료

문자 디바이스 제작방법 - 5

- Read의 return value
  - 요구한 만큼 읽은 경우 count와 같은 값을 리턴
  - 요구한 값보다 작을 경우 count보다 작은 양수 리턴
  - EOP(End of file)인 경우 0을 리턴
  - 에러가 발생하면 음수 리턴
- Write
  - 요구한 만큼 기록한 경우 count와 같은 값을 리턴
  - 요구한 크기보다 적게 쓴 경우 count보다 작은 양수 리턴
  - 하나도 쓰지 못한 경우 0 리턴, write()함수 재시도
  - 에러가 발생한 경우 음수 리턴

DATA 전달 방법

- 함수
  - get_user(void *, Const void *addr)
    - ✍사용자 공간의 어드레스 addr로 부터 변수x로 바이트를 복사
  - put_user(void *, Const void *addr)
    - ✍addr로부터 변수 x까지 사용자 공간으로 바이트를 복사
  - copy_to_user(void *to, void *from, unsinged long size)
  - copy_from_user(void *to, void *from, unsinged long size)
  - <asm/uaccess.h> 참조



문자 디바이스 제작방법 - 6

- ioctl
  - 읽기/쓰기 이외의 부가적인 연산을 위한 인터페이스
  - 디바이스 설정 및 하드웨어 제어 (향상된 문자 드라이버 작성 가능)
  - **ioctl** 시스템 콜은 디바이스에 특화된 명령을 나타내는 방법을 제공한다.
  - 예) reading이나 writing이 아닌 플로피 디스크 트랙을 포맷하는 것과 같은 명령

문자 디바이스 제작방법 - 7

- Ioctl 명령어 생성 인자
  - Type : magic number (참조 Documatation/ioctl-number.txt)
  - Ordinal number : 구현 하고자하는 기능(커맨드)들을 구별
  - Direction of transfer : read only, write only, read-write 구분
    - _IOC_NONE
    - _IOC_READ
    - _IOC_WRITE
    - _IOC_READ | _IOC_WRITE
  - Size of argument : ioctl()의 argument size
  - <include/asm/ioctl.h> 파일과 Documentation/ioct-number.txt 참조

문자 디바이스 제작방법-8

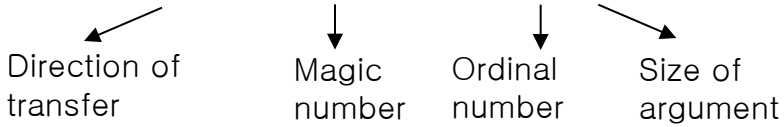
- Ioctl 명령 매크로 - <include/asm/ioctl.h>

```
#define _IO(type,nr)            _IOC(_IOC_NONE, (type), (nr), 0)
#define _IOR(type,nr,size)      _IOC(_IOC_READ, (type), (nr), sizeof(size))
#define _IOW(type,nr,size)      _IOC(_IOC_WRITE, (type), (nr), sizeof(size))
#define _IOWR(type,nr,size)     _IOC(_IOC_READ|_IOC_WRITE, (type), (nr), sizeof(size))
```

  - _IO(type, nr)
  - _IOR(type, nr, sizeof(x))
  - _IOW(type, nr, sizeof(x))
  - _IOWR(type, nr, sizeof(x))

Ioctl() 커맨드 선언

```
#define    IOM_LCD_BASE                 0xbc
#define    IOM_LCD_COMMAND_SET          _IOW(IOM_LCD_BASE,0,32)
#define    IOM_LCD_FUNCTION_SET         _IOW(IOM_LCD_BASE,1,32)
#define    IOM_LCD_DISPLAY_CONTROL      _IOW(IOM_LCD_BASE,2,32)
#define    IOM_LCD_CURSOR_SHIFT         _IOW(IOM_LCD_BASE,3,32)
```

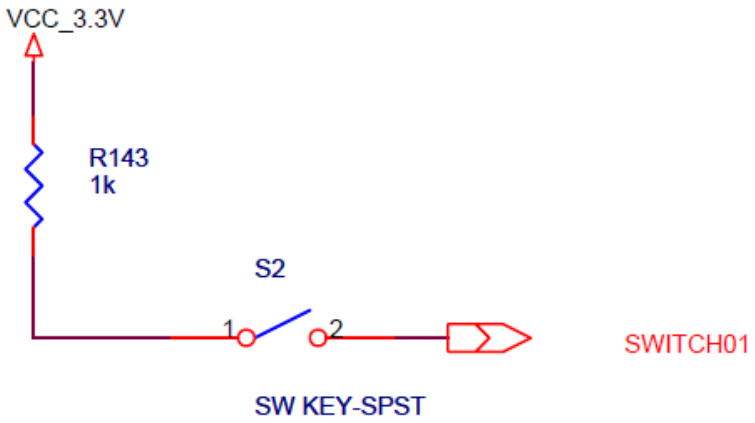


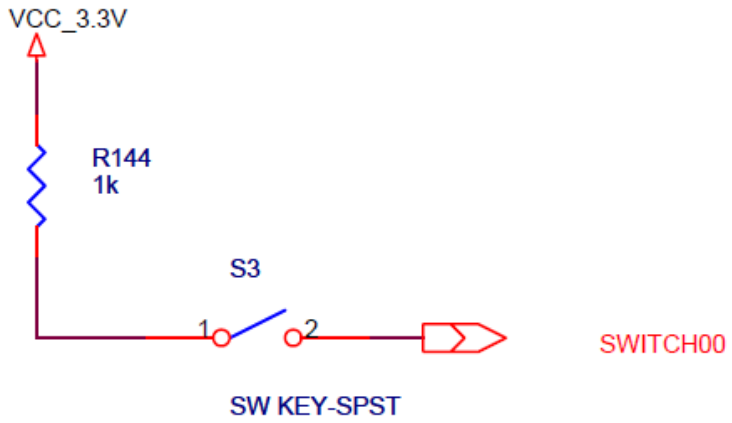

```
int iom_lcd_ioctl(struct inode *inode, struct file *file, unsigned int cmd,unsigned long gdata)
{
    copy_from_user(&strcommand, (char *)gdata, 32);

    switch(cmd){
        case IOM_LCD_COMMAND_SET:        /* xxxx */ break;
        case IOM_LCD_FUNCTION_SET:       /* xxxx */ break;
        case IOM_LCD_DISPLAY_CONTROL:    /* xxxx */ break;
        case IOM_LCD_CURSOR_SHIFT:       /* xxxx */ break;
        default: printk("driver : no such command!\n"); return -ENOTTY;
    }
    return 0;
}
```

Device Driver Example

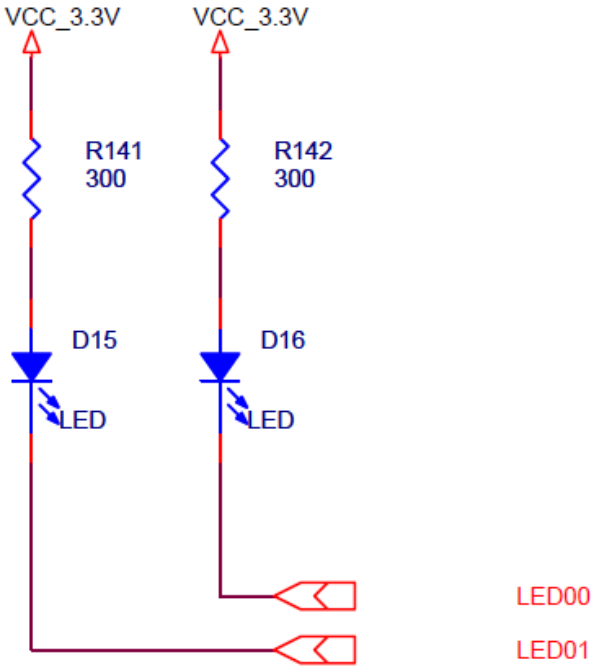
개 요



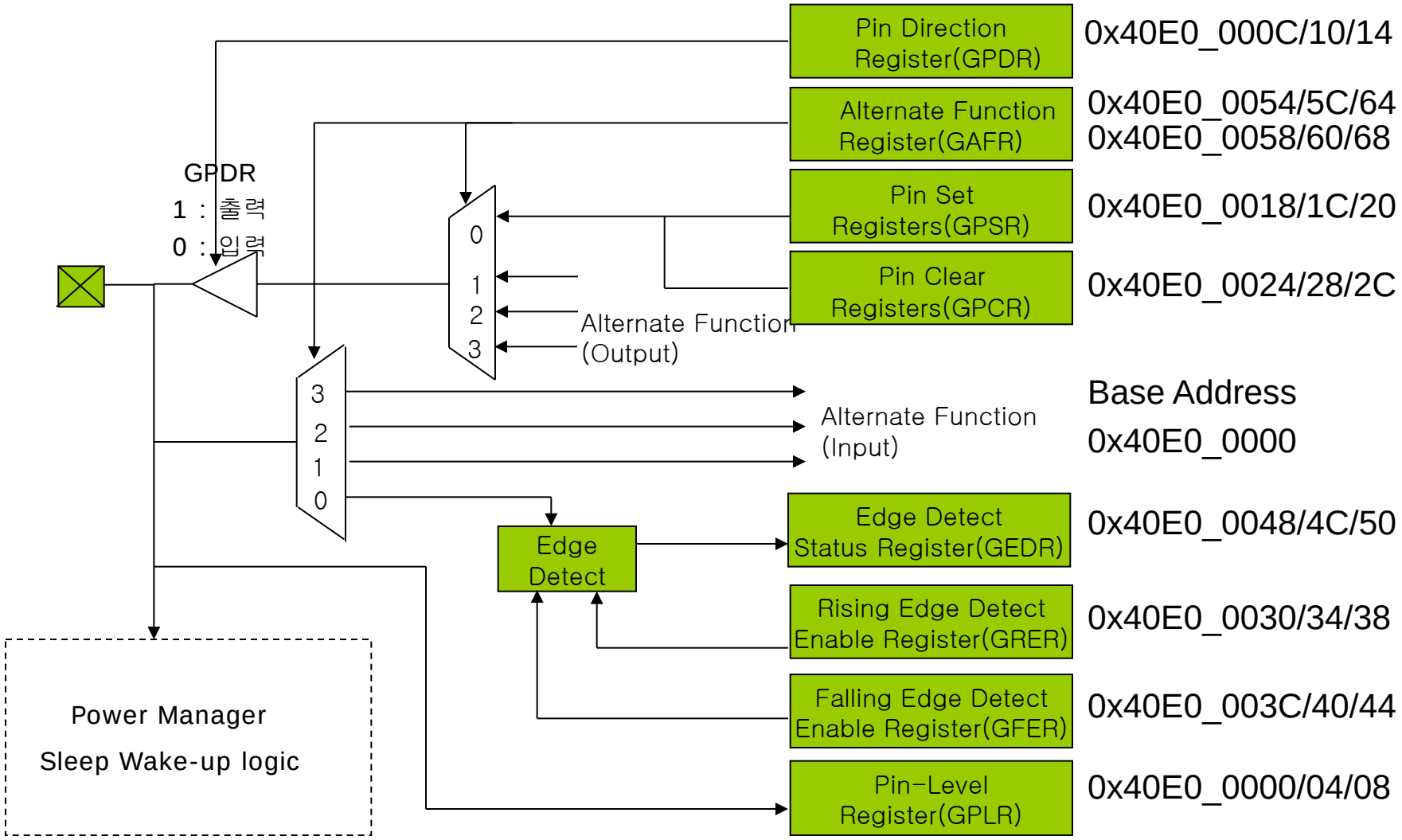


- 개요
  - LED ON-OFF 제어

- GPIO 연결 정보
  - GPIO 53 : push switch
  - GPIO 83 : push switch
  - GPIO 52 : LED
  - GPIO 82 : LED



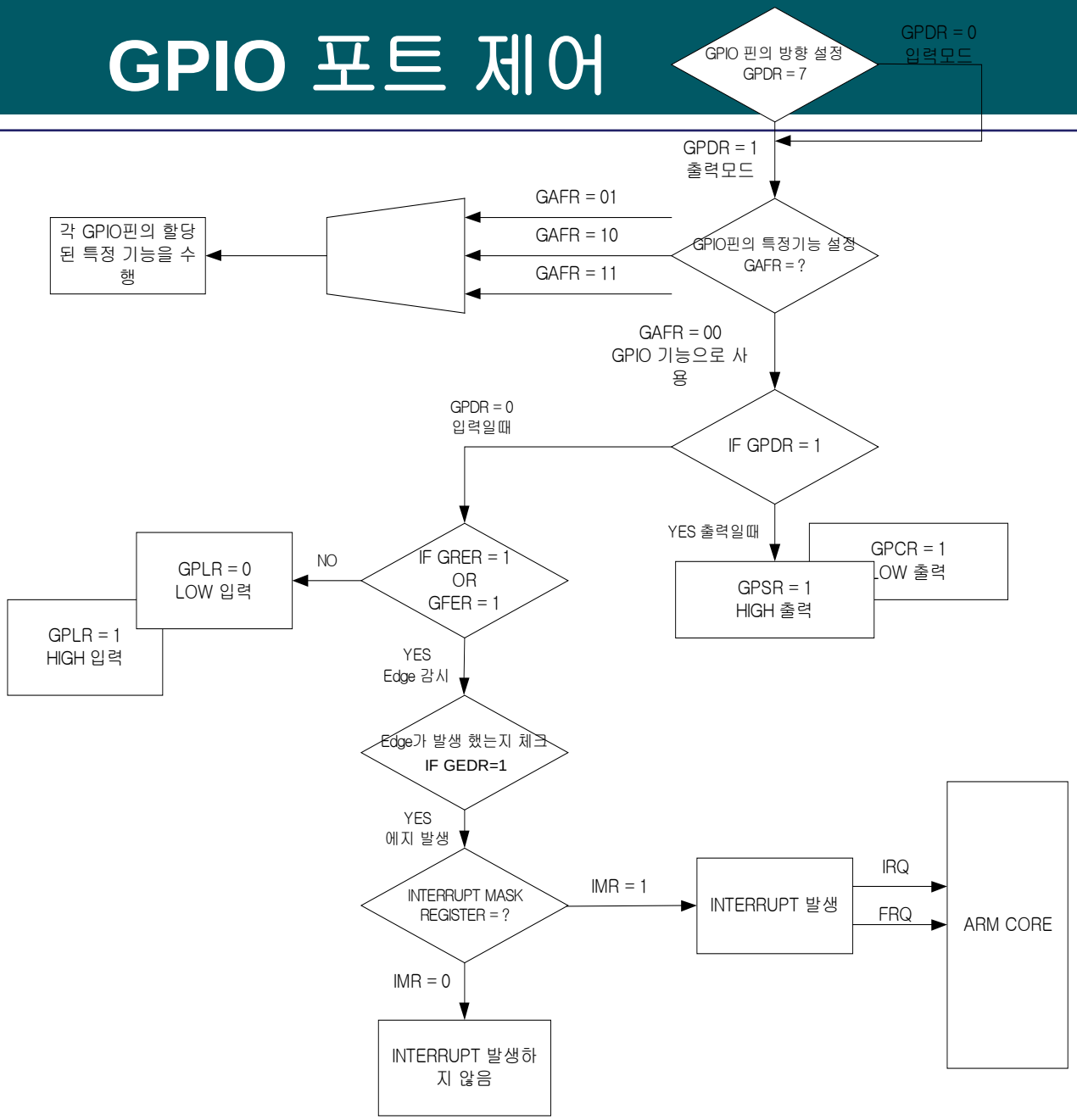
GPIO 레지스터


Pin Direction Register(GPDR)
0x40E0_000C/10/14
Alternate Function Register(GAFR)
0x40E0_0054/5C/64
0x40E0_0058/60/68
Pin Set Registers(GPSR)
0x40E0_0018/1C/20
Pin Clear Registers(GPCR)
0x40E0_0024/28/2C
GPDR
1 : 출력
0 : 입력
0
1
2
3
Alternate Function (Output)
3
2
1
0
Alternate Function (Input)
Base Address
0x40E0_0000
Edge Detect
Edge Detect Status Register(GEDR)
0x40E0_0048/4C/50
Rising Edge Detect Enable Register(GRER)
0x40E0_0030/34/38
Falling Edge Detect Enable Register(GFER)
0x40E0_003C/40/44
Pin-Level Register(GPLR)
0x40E0_0000/04/08
Power Manager
Sleep Wake-up logic

GPIO 레지스터

- GPLR (GPIO Pin-Level Register)
  - GPIO가 input으로 설정 되었을 때 level state를 나타냄.
- GPDR (GPIO Direction Register)
  - GPIO의 input or output 방향 설정
- GPSR (GPIO Set Register)
  - GPIO가 output으로 설정 되었을 때 set pin level high.
- GPCR (GPIO Clear Register)
  - GPIO가 output으로 설정 되었을 때 clear pin level low
- GRER (GPIO Rising Edge Detect Enable Register)
- GFER (GPIO Falling Edge Detect Enable Register)
- GEDR (GPIO Edge Detect Status Register)
- GAFR (GPIO Alternate Function Register)

GPIO 포트 제어


GPIO 핀의 방향 설정
GPDR = 7
GPDR = 0
입력모드
GPDR = 1
출력모드
GPIO핀의 특정기능 설정
GAFR = ?
GAFR = 01
GAFR = 10
GAFR = 11
각 GPIO핀의 할당된 특정 기능을 수행
GAFR = 00
GPIO 기능으로 사용
IF GPDR = 1
GPDR = 0
입력일때
YES 출력일때
GPSR = 1
HIGH 출력
GPCR = 1
LOW 출력
IF GRER = 1
OR
GFER = 1
NO
GPLR = 0
LOW 입력
GPLR = 1
HIGH 입력
YES
Edge 감시
Edge가 발생 했는지 체크
IF GEDR=1
YES
에지 발생
INTERRUPT MASK
REGISTER = ?
IMR = 1
IMR = 0
INTERRUPT 발생
IRQ
FRQ
ARM CORE
INTERRUPT 발생하지 않음

GPIO output 제어 방법

- GPIO 82번을 OUTPUT 방향으로 설정

  GPDR2 |= 0x40000
- GPIO 82번을 GPIO용도로 설정

  GAFR2_U &= ~0x30
- LED를 ON 시키기 위해서는 0을 출력

  GPCR2 |= 0x40000
- LED를 OFF 시키기 위해서는 1을 출력

  GPSR2 |= 0x40000

Table 24-7. GPDR2 Bit Definitions

Physical Address 0x40E0_0014 — **GPDR2** — **GPIO Controller**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
User Settings																																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PD95	PD94	PD93	PD92	PD91	PD90	PD89	PD88	PD87	PD86	PD85	PD84	PD83	PD82	PD81	PD80	PD79	PD78	PD77	PD76	PD75	PD74	PD73	PD72	PD71	PD70	PD69	PD68	PD67	PD66	PD65	PD64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Access	Name	Description
<31:0>	R/W	PD*x*	GPIO Pin Direction 'x' (where x = 64 to 95) This field configures the direction of each GPIO. 0 = Pin configured as an input. 1 = Pin configured as an output.

Table 24-30. GAFR2_U Bit Definitions

Physical Address 0x40E0_0068 — **GAFR2_U** — **GPIO Controller**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
User Settings																																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AF95		AF94		AF93		AF92		AF91		AF90		AF89		AF88		AF87		AF86		AF85		AF84		AF83		AF82		AF81		AF80	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Access	Name	Description
<31:0>	R/W	AFx	GPIO 'x' Alternate Function Select Bits (where x = 80 through 95) A bit-pair in this register determines the corresponding GPIO pin's functionality as one of the alternate functions that is mapped to it or as a generic GPIO pin. 0b00 = The corresponding GPIO pin (GPIO<x>) is used as a general-purpose I/O. 0b01 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 1. 0b10 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 2. 0b11 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 3.

GPIO LED 테스트 프로그램 소스

```
void usrsignal(int sig)
{               STOP_LED = ~STOP_LED;                   }
int main(void)
{
int fd,id;
unsigned char c;

                while(1){
                        fd = open("/dev/led_gpio",O_WRONLY);
                        if (fd < 0){    printf("Device Open Error\n");
                                exit(1);
                        }
/* kernel/signal.c */   (void)signal(SIGUSR1,usrsignal);    /* SIGUSR1 시그널이 들어오면 usrsignal()을 실행 */
                        id = getpid();
                        ioctl(fd,GPIO_ID_SET,&id,4);
                        for(;;){        if(STOP_LED) break;         /* STOP_LED=1 이면 LED stop */
                                c = GPIO82_ON;
                                write(fd,&c,1);                 /* GPIO82 LED ON */
                                usleep(10);                     /* delay */
                                c = GPIO82_OFF;
                                write(fd,&c,1);                 /* GPIO82 LED OFF */
                                usleep(10);
                                c = GPIO52_ON;
                                write(fd,&c,1);
                                usleep(10);
                                c = GPIO52_OFF;
                                write(fd,&c,1);
                                usleep(10);
                        }
                        close(fd);
                }
}
```

디바이스 드라이버 매크로 선언부

```
/*
 * The name for our device, as it will appear in /proc/devices
 */

#define  DEVICE_NAME   "led_gpio"  // LED 디바이스 명
#define  LED_MAJOR253          // LED 디바이스 메이저 번호
#define  GPIO82_ON  0x01
#define  GPIO52_ON  0x02
#define  GPIO82_OFF0x10
#define  GPIO52_OFF0x20
#define  ALL_ON                    0x03
#define  ALL_OFF      0x30

#define LED_GPIO_BASE  0xbc          //LED IOCTL 명령 magic number
#define GPIO_ID_SET  _IOW(LED_GPIO_BASE,0,4)
```

드라이버 소스 - module_init()

```
static int led_gpio_init(void)
{
 /** Register the character device  */
    int return_val1, return_val2;
    Major = register_chrdev (LED_MAJOR, DEVICE_NAME, &led_fops);
    if (Major < 0) {
                        printk ("LED init_module: failed with %d\n", LED_MAJOR);
                        return Major;
    }
    printk ("LED driver registred: major = %d\n", LED_MAJOR);

   /**  GPIO52,GPIO82을 출력 으로 설정한다.  */
    GPDR1 |= 1 << 20;
    GPDR2 |= 1 << 18;
   /**  GPIO53, GPIO83을 입력 으로 설정한다.  */
    GPDR1 &= ~(1 << 21);
    GPDR2 &= ~(1 << 19);

    set_irq_type( IRQ_GPIO(53), IRQT_FALLING );
    set_irq_type( IRQ_GPIO(83), IRQT_FALLING );
    return_val1 = request_irq(IRQ_GPIO(53), pushsw_handler1, SA_INTERRUPT, "PUSH SW 53 INT", NULL );
    return_val2 = request_irq(IRQ_GPIO(83), pushsw_handler2, SA_INTERRUPT, "PUSH SW 83 INT", NULL );

    if( return_val1 < 0 )
    {
                        printk(KERN_ERR "Acumen270_push_init() : Can't reqeust irq %#010x\n", IRQ_GPIO(53) );
                        return -1;
    }
    if( return_val2 < 0 )
    {
                        printk(KERN_ERR "Acumen270_push_init() : Can't reqeust irq %#010x\n", IRQ_GPIO(83) );
                        return -1;
    }
    return 0;
}
```

드라이버 소스 - module_exit()

```
/*
 * exit - unregister the appropriate file from /proc
 */

void led_gpio_exit(void)
{
		int ret;

		/*
		 * Unregister the device
		 */
		ret = unregister_chrdev (LED_MAJOR, DEVICE_NAME);
		if (ret < 0) {
				printk ("unregister_chrdev: error %d\n", ret);
		}
		else {
				printk ("module clean up ok!! \n");
		}

		free_irq (IRQ_GPIO(53), NULL);
		free_irq (IRQ_GPIO(83), NULL);
}
```

드라이버 소스 - file operations

```
static struct file_operations led_fops =
{
        open:           led_gpio_open,
        write:          led_gpio_write,
        release:        led_gpio_release,
        ioctl:          led_gpio_ioctl,
};
```

```
static struct file_operations led_fops =
{
        .open =         led_gpio_open,
        .write =        led_gpio_write,
        .release =      led_gpio_release,
        .ioctl =        led_gpio_ioctl,
};
```

드라이버 소스 - open(), release()

```
int led_gpio_open (struct inode *inode, struct file *file)
{
        /*
         * Get major / minor numbers when needed
         */
        if (Device_Open) {
                return -EBUSY;
        }
        Device_Open++;

        return 0;
}

int led_gpio_release (struct inode *inode, struct file *file)
{
        /* We're now ready for our next caller */
        Device_Open--;
        return 0;
}
```

드라이버 소스- led_gpio_wirte

```
ssize_t led_gpio_write (struct file *file,
                        const char *buffer,/* buffer */
                        size_t length,          /* length of buffer */
                        loff_t * offset)        /* offset in the file */
{
        const char *tmp = buffer;
        unsigned char c=0;

        get_user( c, (unsigned char *)(tmp) );
        switch(c){
                case GPIO82_ON      : GPCR2 |= 1 << 18; break;  // GPIO82 ON
                case GPIO52_ON      : GPCR1 |= 1 << 20; break;  // GPIO52 ON
                case GPIO82_OFF     : GPSR2 |= 1 << 18; break;  // GPIO82 OFF
                case GPIO52_OFF     : GPSR1 |= 1 << 20; break;  // GPIO52 OFF
                case ALL_ON         : GPCR2 |= (1 << 18);
                                      GPCR1 |= (1 << 20);
                                      break;                    // GPIO82,52 ON
                case ALL_OFF        : GPSR2 |= (1 << 18);
                                      GPSR1 |= (1 << 20);
                                      break;                    // GPIO82,52 OFF
                default             : printk("\n LED Write Error "); break;
        }
        return (length);
}
```

드라이버 소스- led_gpio_ioctl

```
static int led_gpio_ioctl(struct inode *inode, struct file *file,
                                        unsigned int cmd, unsigned long gdata)
{
        int id;                                         /* gdata is application PID */
        copy_from_user(&id,(char *)gdata,4);  /* copy from gdata to id */

        switch(cmd){
                case GPIO_ID_SET :
                        app_id = id;      /* for use in interrupt handler */
                        break;
                default : break;
        }
        return 0;
}
```

```
ioctl(fd,GPIO_ID_SET,&id,4)
```

드라이버 소스 - pushsw_handler()

```
static irqreturn_t pushsw_handler1(int irq, void *dev_id, struct pt_regs *regs )
{
        printk("\n GPIO53 Push SW Detect");
        kill_proc(app_id,SIGUSR1,1);

        return IRQ_HANDLED;
}

static irqreturn_t pushsw_handler2(int irq, void *dev_id, struct pt_regs *regs )
{
        printk("\n GPIO83 Push SW Detect");
        kill_proc(app_id,SIGUSR1,1);

        return IRQ_HANDLED;
}
```

디바이스 드라이버 커널에 포함시키기

- 개요
  1. 디바이스 드라이버 커널소스에 복사
  2. 관련 Makefile 수정
  3. 관련 Kconfig 파일 수정
  4. 커널 컴파일시 드라이버 선택
  5. 커널 컴파일

디바이스 드라이버 커널에 포함시키기-1

- 1. 디바이스 드라이버 복사
  - # cd /root/acumen/led-key/
  - # cp ./led_driver.c /root/acumen/kernel/linux-2.6.9-acumen-v1.4 /drivers/char/
- 2. 관련 Makefile 수정
  - # cd /root/acumen/kernel/linux-2.6.9-acumen-v1.4/drivers/char/
  - # vi Makefile
    - obj-$(CONFIG_LED)+=led_driver.o 포함

```
obj-$(CONFIG_SGI_SNSC)      += snsc.o
obj-$(CONFIG_MMTIMER)       += mmtimer.o
obj-$(CONFIG_VIOCONS)       += viocons.o
obj-$(CONFIG_VIOTAPE)       += viotape.o
obj-$(CONFIG_HVCS)          += hvcs.o

obj-$(CONFIG_LED)           += led_driver.o
```

디바이스 드라이버 커널에 포함시키기-2

- 3. 관련 Kconfig 파일 수정
  - # cd /root/acumen/kernel/linux-2.6.9-acumen-v1.4/drivers/char/
  - # vi Kconfig
    - bool “ LED GPIO ” 추가

```
config MMTIMER
        tristate "MMTIMER Memory mapped RTC for SGI Altix"
        depends on IA64_GENERIC || IA64_SGI_SN2
        default y
        help
          The mmtimer device allows direct userspace access to the
          Altix system timer.

config  LED
        bool  “ LED GPIO ”

endmenu
```

디바이스 드라이버 커널에 포함시키기-4

- 4. 커널 컴파일 시 led_gpio 드라이버 선택
  - # cd /root/acumen/kernel/linux-2.6.9-acumen-v1.4/
  - # make menuconfig
    - Character devices → LED GPIO 체크
  - # make zImage

- 생성된 커널 위치
  - /root/acumen/kernel/linux-2.6.9-acumen-v1.4/arch/arm/boot/zImage

- 5. 커널에 포함한 드라이버 타겟에서 테스트
  - 생성된 커널 이미지 타겟 보드에 퓨징
  - 테스트 프로그램 다운로드 / 실행