Module 12: LED 제어 디바이스 드라이버

ESP30076 임베디드 시스템 프로그래밍 (Embedded System Programming)

조 윤 석

전산전자공학부

주차별 목표

- 하드웨어 제어를 위한 디바이스 드라이버 작성 방법 알아보기

- LED 제어용 디바이스 드라이버 작성하기
  - ioremap을 이용한 LED 제어 디바이스 드라이버
  - mmap을 이용한 LED 제어 디바이스 드라이버

디바이스 구분

- 문자 디바이스 (Character device)
  - 자료의 순차성을 지닌 장치로 버퍼를 사용하지 않고 바로 읽고 쓸 수 있는 장치
  - 직렬 포트, 병렬 포트, 마우스, PC 스피커, 터미널 등
- 블록 디바이스 (Block device)
  - 버퍼 캐시(cache)를 통해 블록 단위로 입출력되며, 랜덤 액세스가 가능하고, 파일 시스템을 구축할 수 있음
  - 플로피 디스크, 하드 디스크, CD-ROM, RAM 디스크 등
- 네트워크 디바이스 (Network device)
  - 네트워크 통신을 통해 네트워크 패킷을 주고 받을 수 있는 디바이스
  - Ethernet, PPP, ATM, ISDN, NIC (Network Interface Card) 등

디바이스 드라이버 작성하려면

- 하드웨어에 대한 분명한 이해가 있어야 함
- 소프트웨어 구조에 대한 이해
- 예를 들어 직렬 디바이스 (UART)에 대한 device driver를 작성한다면 다음의 사항들을 분명히 알아야 함 (일부 나열)
  - UART는 세 종류의 레지스터를 가지고 있음
    - Data registers, control registers, status registers
  - 하나의 디바이스 주소에 하나 이상의 디바이스 레지스터들이 있을 수 있음
  - 디바이스는 control register의 bit들을 설정함으로써 초기화 하거나, 설정을 변경할 수 있음
  - 디바이스는 control register의 bit들을 리셋 함으로써 close하거나 리셋을 할 수 있음

디바이스 드라이버 작성하려면

- Control register 비트들은 UART의 모든 동작을 제어할 수 있음
  - 따라서 control register의 각 비트들의 목적 정확히 알아야 함
- Status register 비트들은 디바이스의 현재 상태 (status)에 대한 정보를 가지고 있고, action이 일어날때 마다 해당 플래그의 값들이 변경됨
  - (예)TRH 버퍼 레지스터의 내용이 모든 전송된 후 새로 전송할 비트가 생긴다고 할 경우, 두 상태 사이에 transmitter empty flag의 설정이 변함
  - Status register에서 각 status flag의 목적이 무엇인지 정확히 알아야 함
- 각 레지스터에 대한 주소를 알아야 함
  - 예를 들어 IBM PC의 경우
    - Timer : 0x0040 ~ 0x005F
    - Serial COM1: 0x03F8 ~ 0x03FF
    - Serial COM2: 0x02F8 ~ 0x02FF

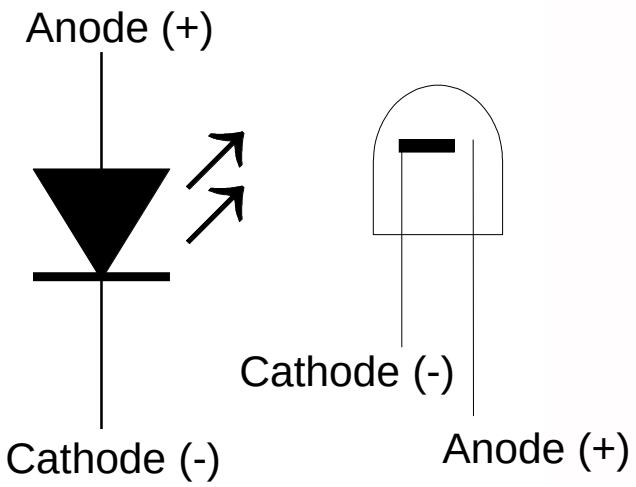
file_operations 구조체

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, 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 (*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 (*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);
};
```

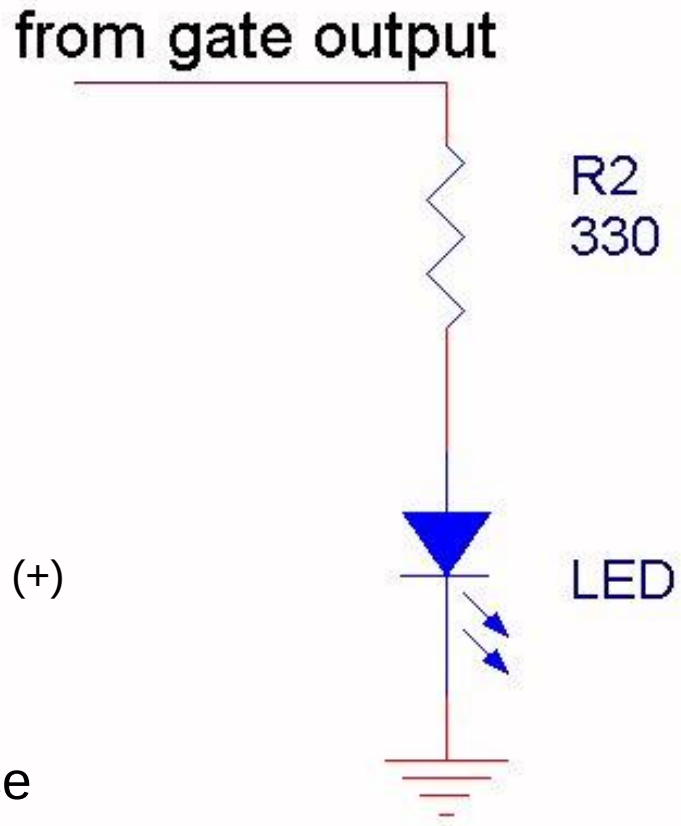
Device Driver에서 사용하는 함수들

- open()
  - 해당 디바이스에 연산을 가하기 위해 해당 디바이스 파일을 열기 위한 함수. 사용수 증가
- read()
  - 해당 디바이스로부터 데이터를 얻은 데이터를 커널 영역에서 사용자 영역으로 복사하기 위한 함수
- write()
  - 사용자 영역의 데이터를 커널 영역으로 복사하기 위한 함수
- release()
  - 해당 드라이버가 응용프로그램에 의해 닫힐 때 호출 하는 함수. 사용수 감소
- ioctl()
  - 읽기 / 쓰기 이외의 부가적인 연산을 위한 인터페이스 - 디바이스 설정 및 하드웨어 제어(향상된 문자드라이버 작성가능)

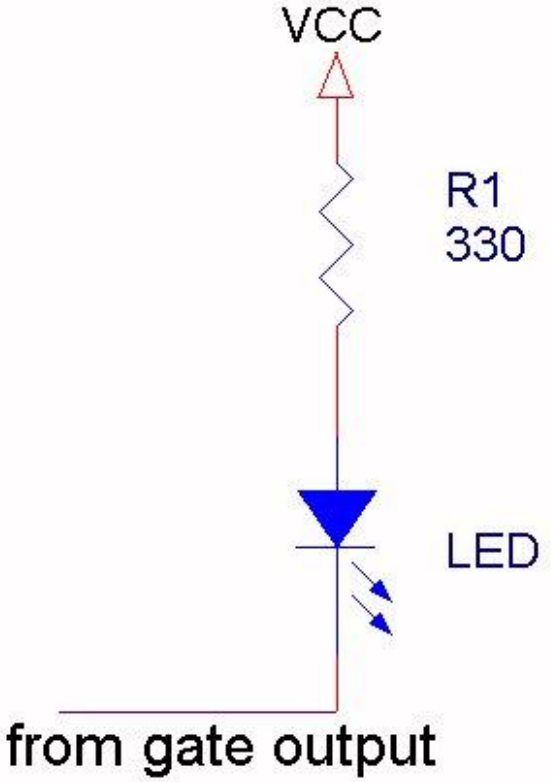
LED



Symbol Appearance

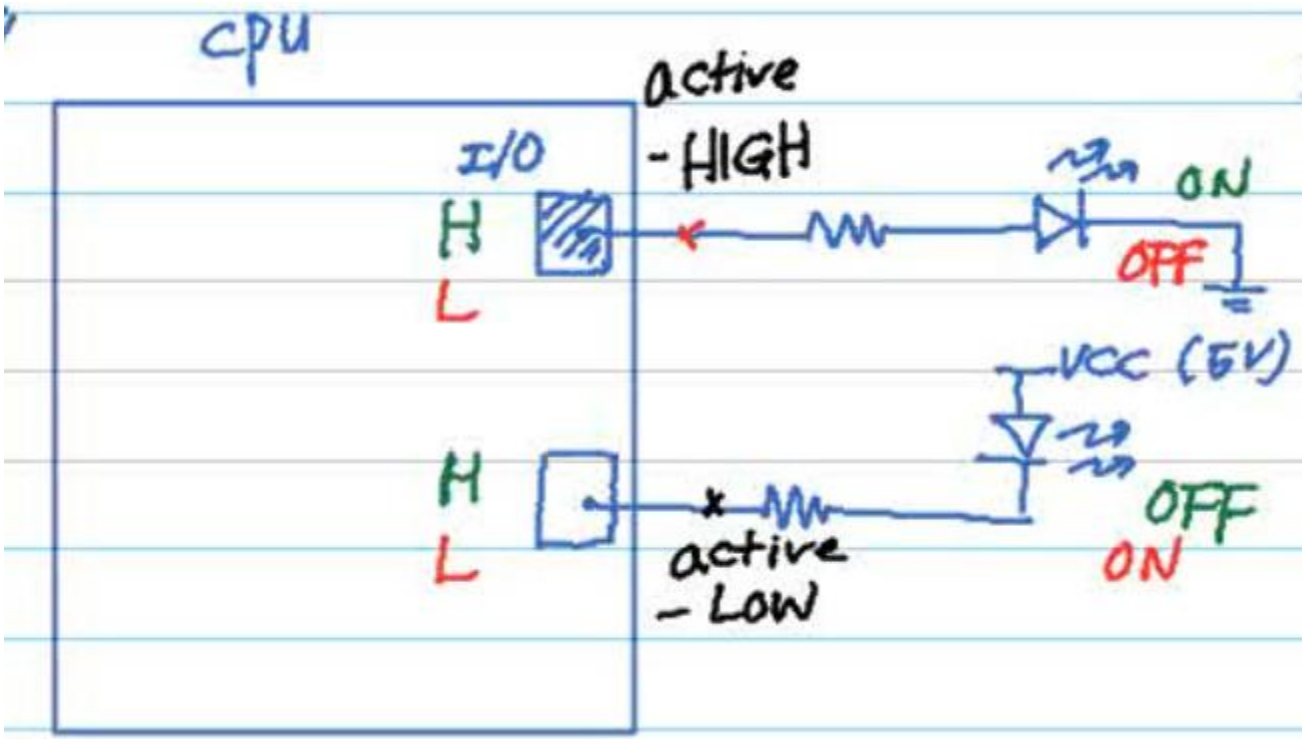


Active-HIGH



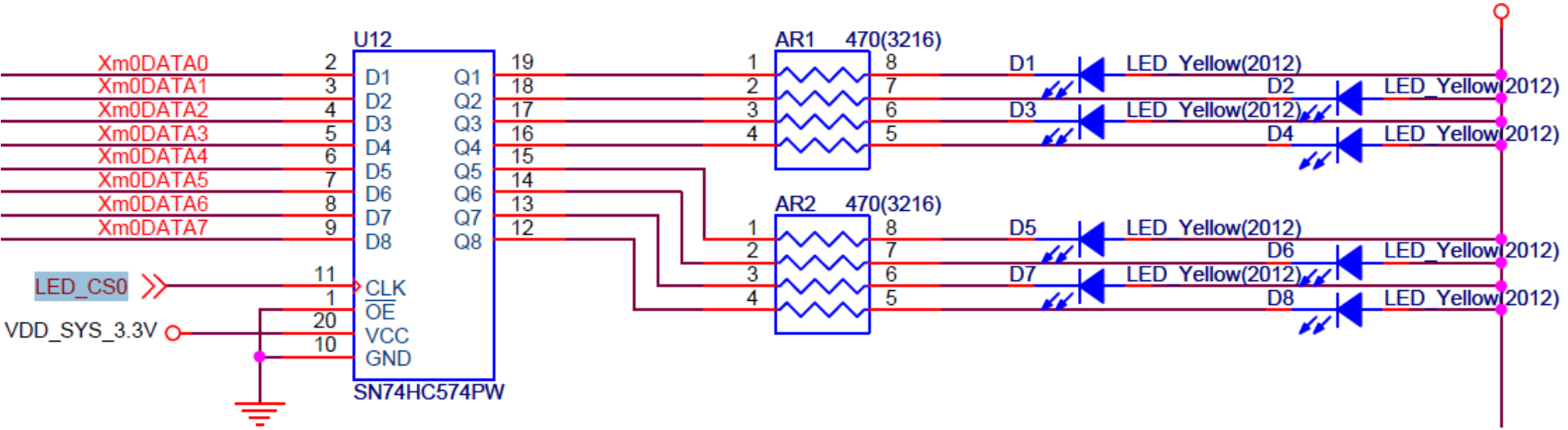
Active-LOW

연결도 파악

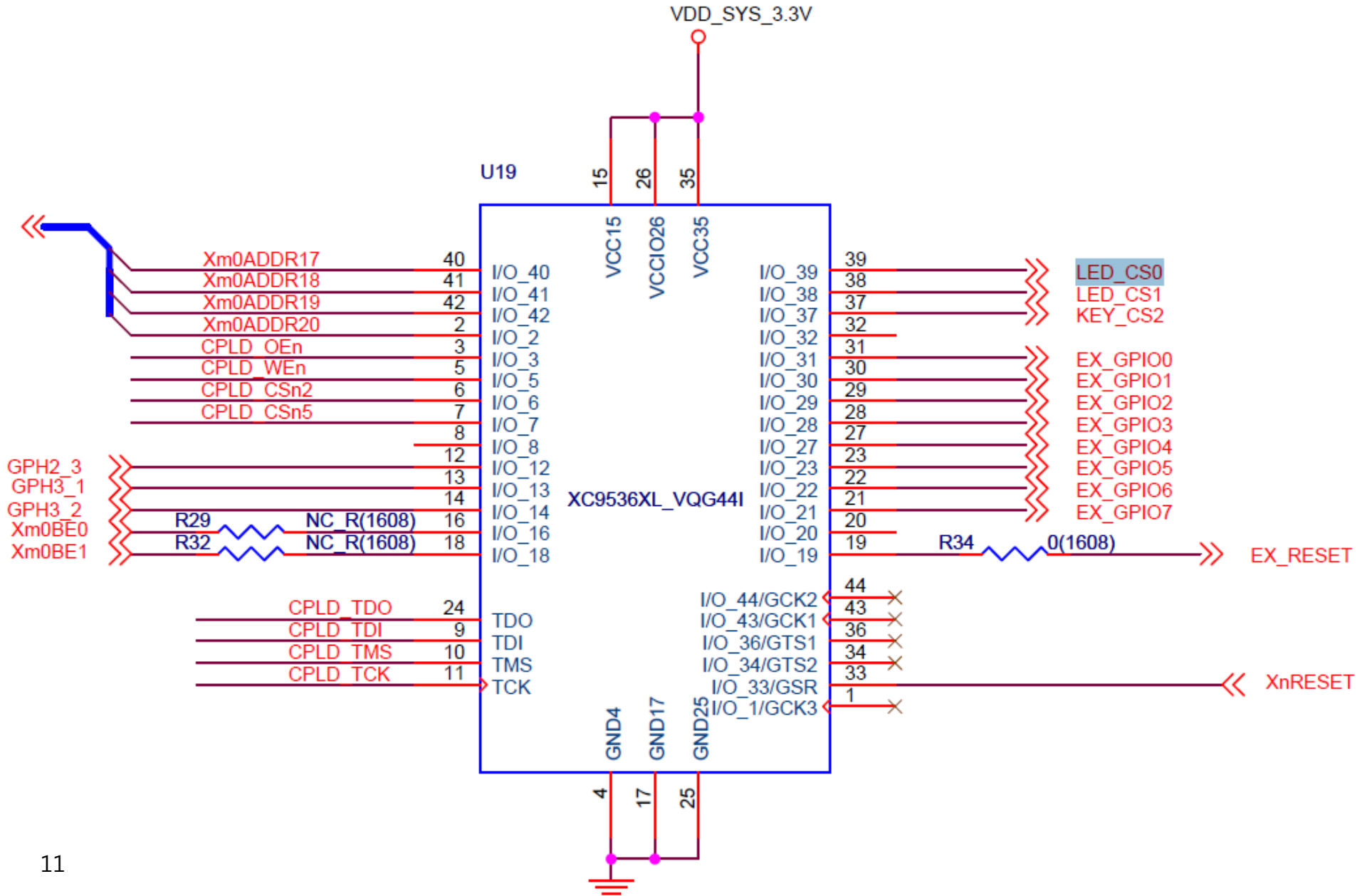


회로도에서 LED 연결 부분 찾기

❑ 확장보드(PC100)에 있는 8개 LED 구동



LED_CS0


VDD_SYS_3.3V
U19
15
26
35
VCC15
VCCIO26
VCC35
Xm0ADDR17
Xm0ADDR18
Xm0ADDR19
Xm0ADDR20
CPLD_OEn
CPLD_WEn
CPLD_CSn2
CPLD_CSn5
40
41
42
2
3
5
6
7
8
I/O_40
I/O_41
I/O_42
I/O_2
I/O_3
I/O_5
I/O_6
I/O_7
I/O_8
GPH2_3
GPH3_1
GPH3_2
Xm0BE0
Xm0BE1
12
13
14
16
18
I/O_12
I/O_13
I/O_14
I/O_16
I/O_18
R29
R32
NC_R(1608)
NC_R(1608)
XC9536XL_VQG44I
I/O_39
I/O_38
I/O_37
I/O_32
I/O_31
I/O_30
I/O_29
I/O_28
I/O_27
I/O_23
I/O_22
I/O_21
I/O_20
I/O_19
39
38
37
32
31
30
29
28
27
23
22
21
20
19
LED_CS0
LED_CS1
KEY_CS2
EX_GPIO0
EX_GPIO1
EX_GPIO2
EX_GPIO3
EX_GPIO4
EX_GPIO5
EX_GPIO6
EX_GPIO7
R34
0(1608)
EX_RESET
CPLD_TDO
CPLD_TDI
CPLD_TMS
CPLD_TCK
24
9
10
11
TDO
TDI
TMS
TCK
I/O_44/GCK2
I/O_43/GCK1
I/O_36/GTS1
I/O_34/GTS2
I/O_33/GSR
I/O_1/GCK3
44
43
36
34
33
1
XnRESET
GND4
GND17
GND25
4
17
25

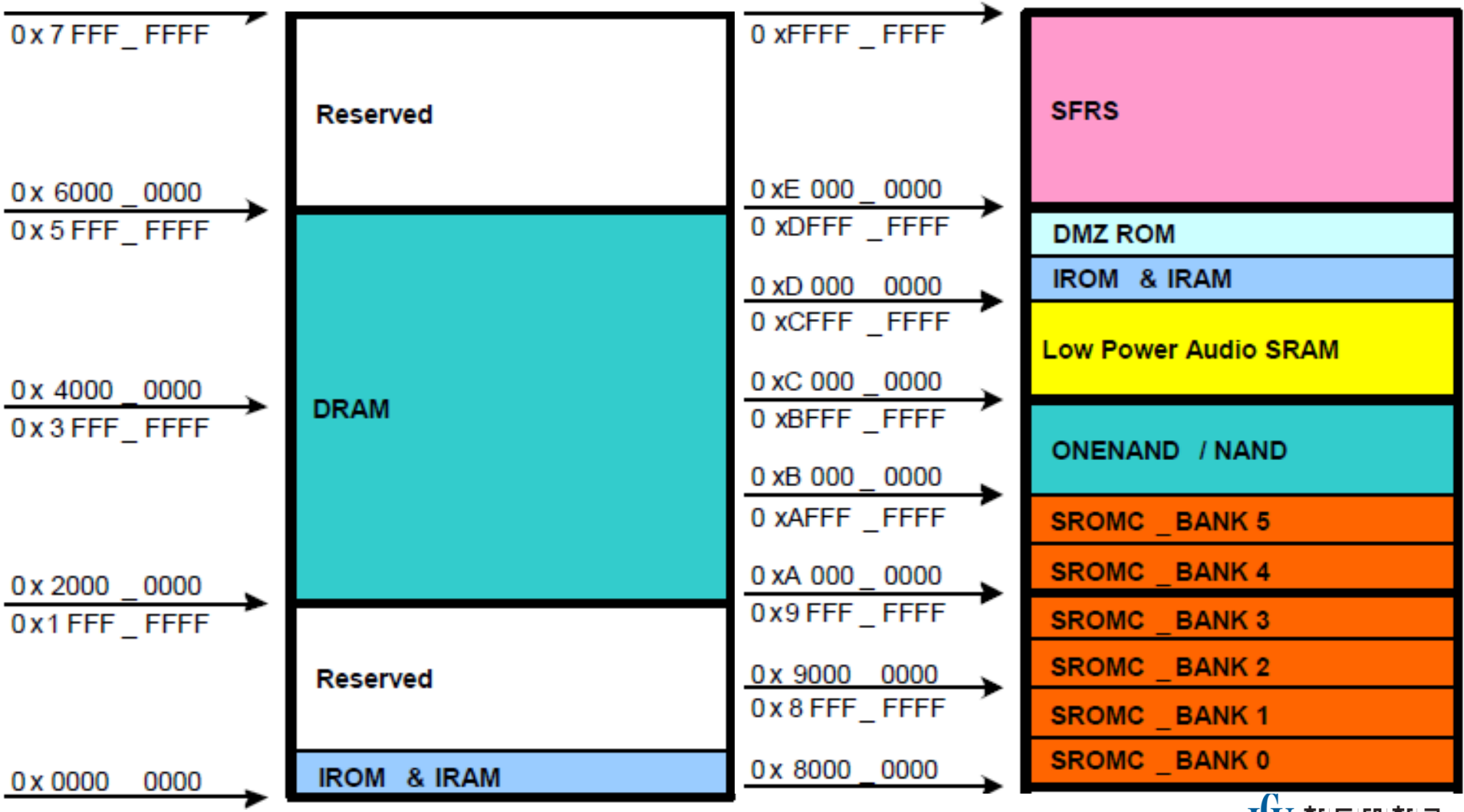
VHDL 코드

```
entity PC100_Decode is
Port ( Xm0ADDR : in STD_LOGIC_VECTOR (20 downto 17);
...
CPLD_CS5 : in STD_LOGIC;
LED_CS0 : out STD_LOGIC;
LED_CS1 : out STD_LOGIC;
KEY_CS2 : out STD_LOGIC;
...
end PC100_Decode;
architecture Behavioral of PC100_Decode is
begin
LED_CS0 <= '0' when (XnRESET = '1' and CPLD_CS5 = '0' and Xm0ADDR(20
downto 17) = "0000") else '1';
LED_CS1 <= '0' when (XnRESET = '1' and CPLD_CS5 = '0' and Xm0ADDR(20
downto 17) = "0001") else '1';
KEY_CS2 <= '0' when (XnRESET = '1' and CPLD_CS5 = '0' and Xm0ADDR(20
downto 17) = "0010") else '1';
end Behavioral;
```

CPLD 회로

- 물리주소 0xA000_0000를 가상주소로 변환
  - unsigned short *srom_bank5;
  - srom_bank5 = ioremap(0xA8000000, 0x1000);
  - *(srom_bank5) 의 형태로 물리 주소에 접근
- 데이터에 값 쓰기
  - *(srom_bank5) = 0xAAAA; // 1010_1010_1010_1010(2)
  - 물리주소 0xA8000000 에 0xAAAA가 쓰여짐
    - AP(PV210)에 의해서 Xm0CSn5 가 LOW로 떨어짐
    - 물리주소 0xA8000000을 통해서 0xAAAA 데이터를 받은 SROM 컨트롤러는 Xm0ADDR[15:0]에 0x0000를 출력하고 Xm0DATA[15:0] 에 0xAAAA 를 출력

Memory Address Map



Device Specific Address Space

2.1.1 DEVICE SPECIFIC ADDRESS SPACE

Address		Size	Description
0x0000_0000	0x1FFF_FFFF	512MB	Boot area
0x2000_0000	0x3FFF_FFFF	512MB	DRAM 0
0x4000_0000	0x5FFF_FFFF	512MB	DRAM 1
0x8000_0000	0x87FF_FFFF	128MB	SROM Bank 0
0x8800_0000	0x8FFF_FFFF	128MB	SROM Bank 1
0x9000_0000	0x97FF_FFFF	128MB	SROM Bank 2
0x9800_0000	0x9FFF_FFFF	128MB	SROM Bank 3
0xA000_0000	0xA7FF_FFFF	128MB	SROM Bank 4
0xA800_0000	0xAFFF_FFFF	128MB	SROM Bank 5

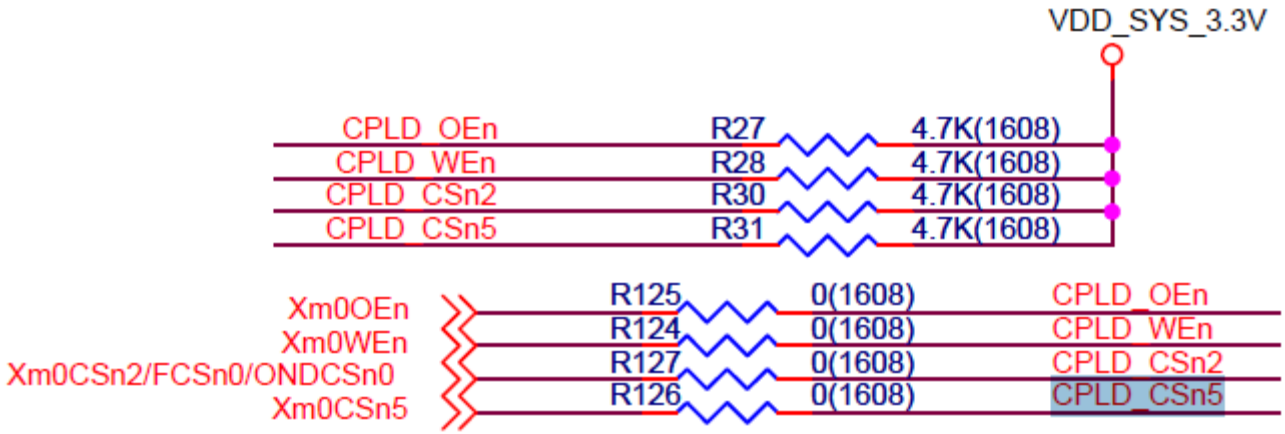
(Ref) S5PV210 CPU User Manual (S5PV210_Usernamual_Rev1.0.pdf) p.23~24

SROM Interface

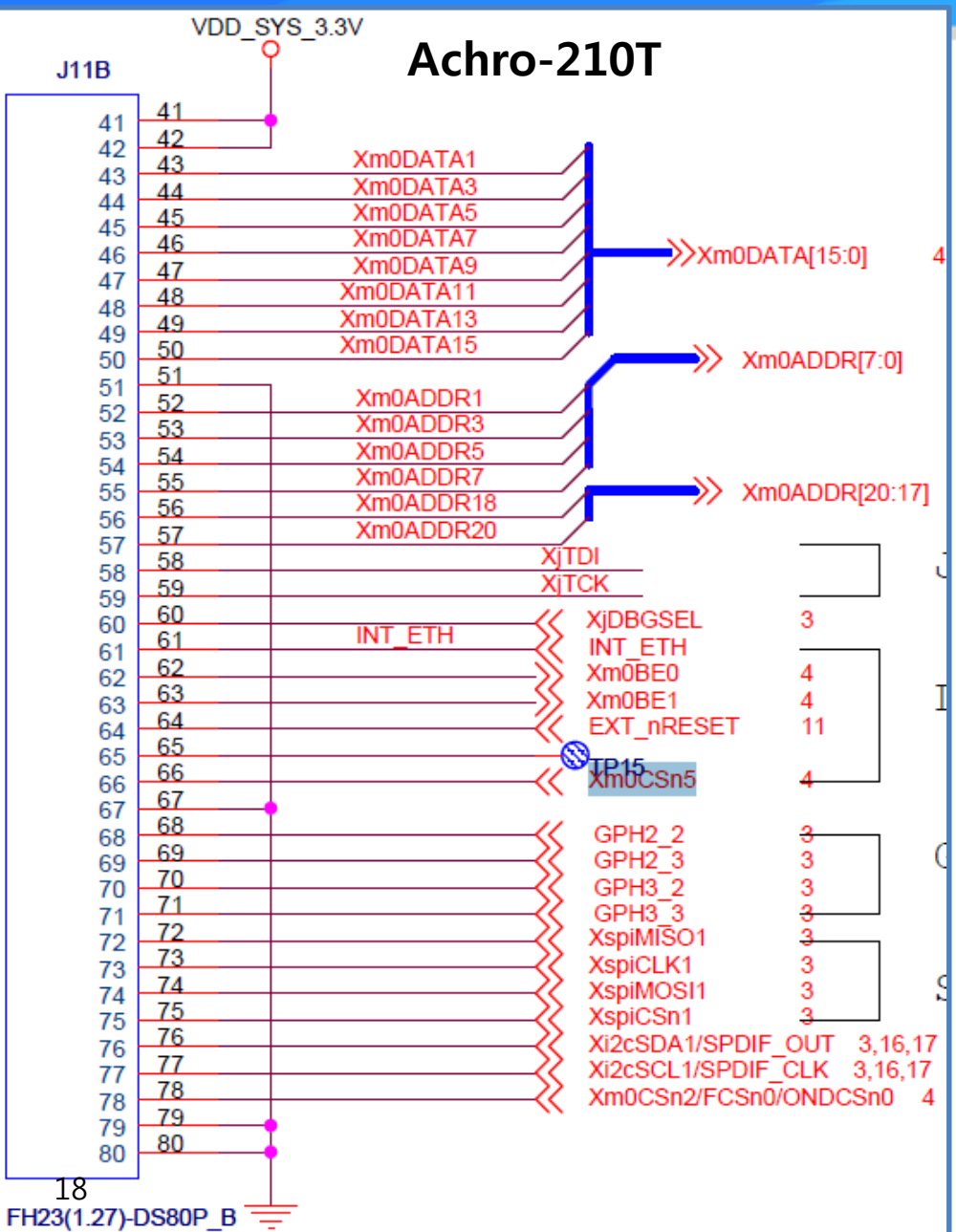
- SROM 인터페이스에 할당된 특정 주소 접근시
  - AP(Application Processor, PV210) 내부에 존재하는 SROM 컨트롤러가 칩 셀렉트 동작을 자동으로 수행

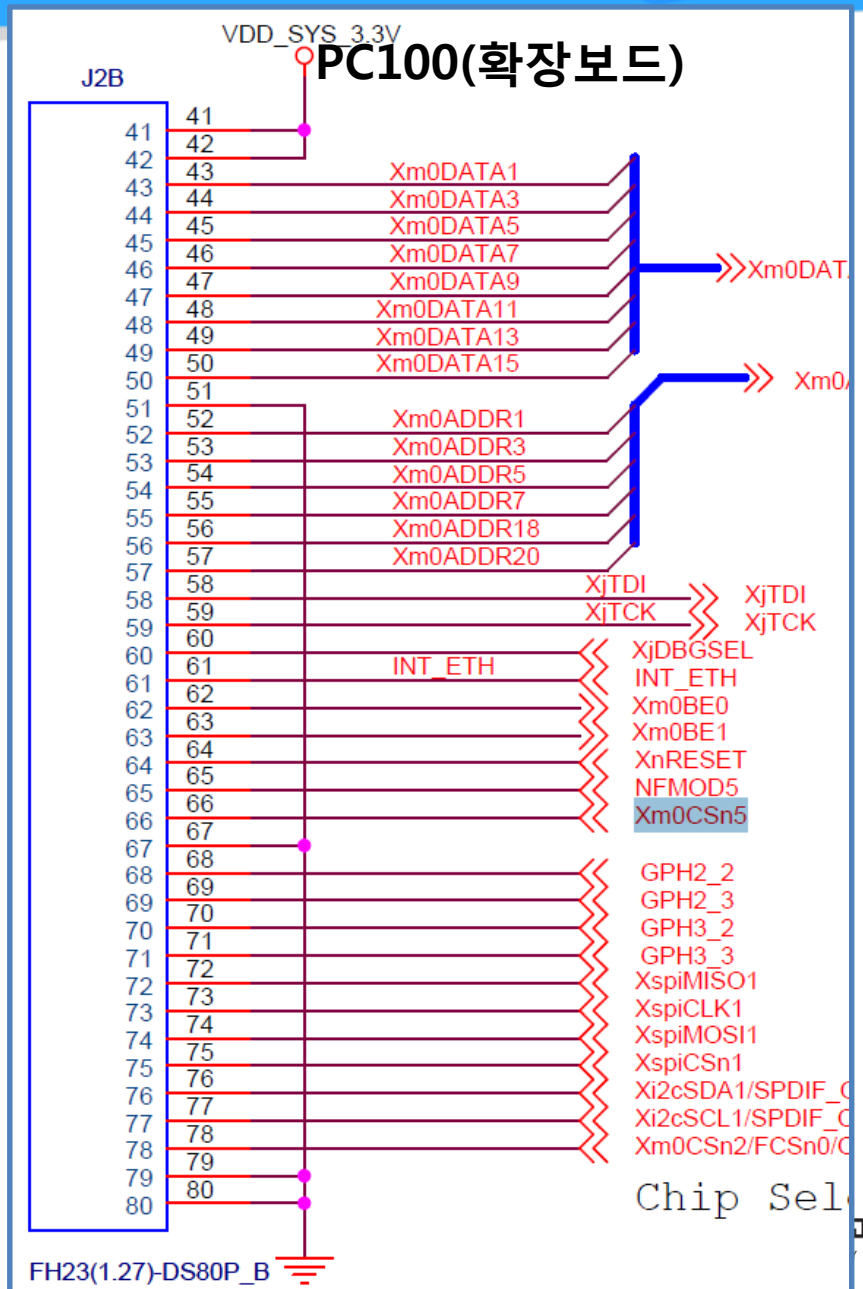
구 분	시작 주소	마지막 주소	크기	동작하는 CS
SROM Bank 0	0x8000_0000	0x87FF_FFFF	128MB	Xm0CSn0
SROM Bank 1	0x8800_0000	0x8FFF_FFFF	128MB	Xm0CSn1
SROM Bank 2	0x9000_0000	0x97FF_FFFF	128MB	Xm0CSn2
SROM Bank 3	0x9800_0000	0x9FFF_FFFF	128MB	Xm0CSn3
SROM Bank 4	0xA000_0000	0xA7FF_FFFF	128MB	Xm0CSn4
SROM Bank 5	0xA800_0000	0xAFFF_FFFF	128MB	Xm0CSn5

CPLD_CSn5 / Xm0CSn5

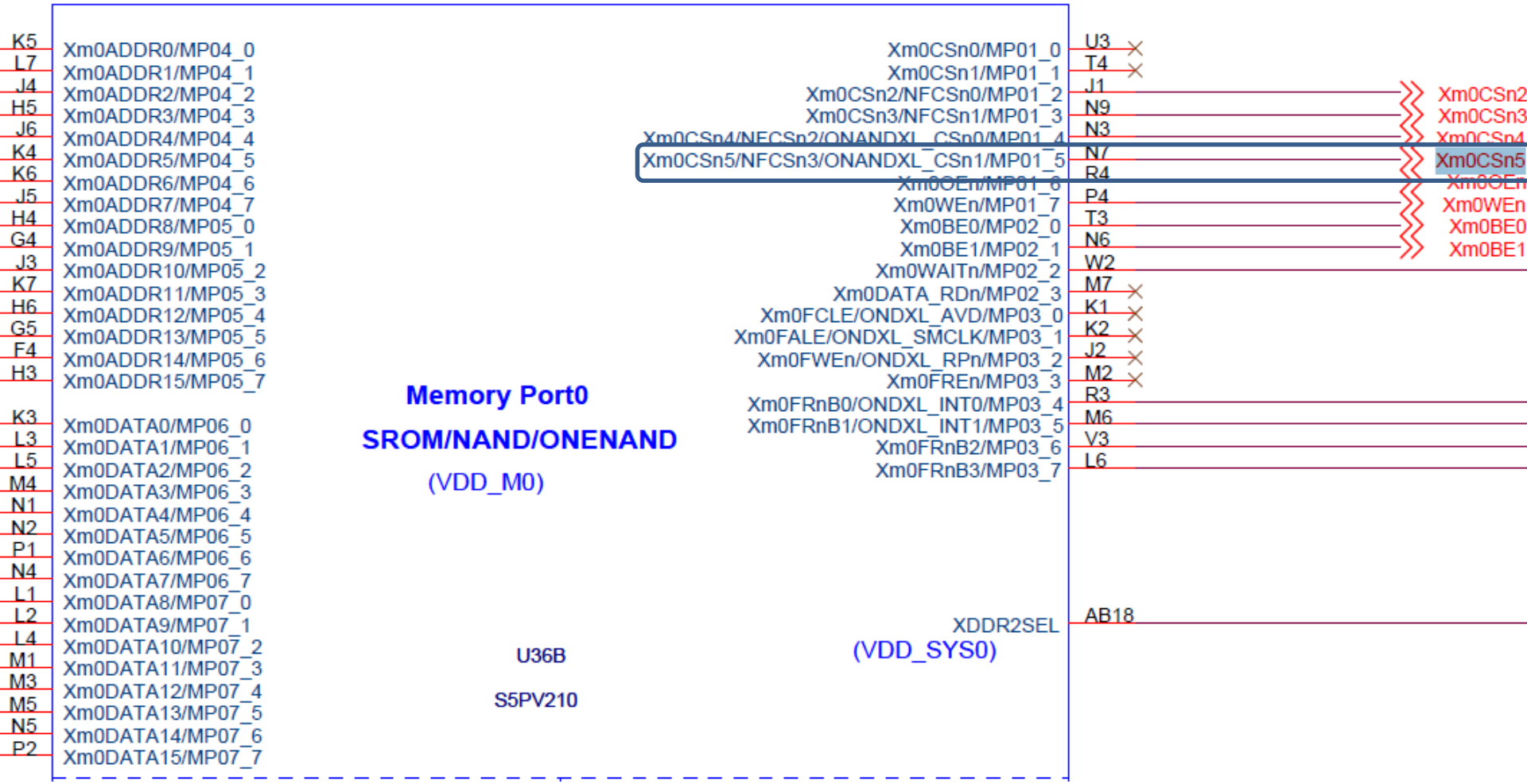


Xm0CSn5 in PC100





Xm0CSn5 in S5PV210(CPU)



Xm0CSn5

- Memory port 0

Pin Name	Func0		Func1		Func2		Func3		Default	Reset
	Signal	IO	Signal	IO	Signal	IO	Signal	IO		
XM0CSN_0	SROM_CSn[0]	O							Func0	O(H)
XM0CSN_1	SROM_CSn[1]	O							Func0	O(H)
XM0CSN_2	SROM_CSn[2]	O	NFCSn[0]	O					Func1	O(H)
XM0CSN_3	SROM_CSn[3]	O	NFCSn[1]	O					Func1	O(H)
XM0CSN_4	SROM_CSn[4]	O	NFCSn[2]	O			ONANDXL_CSn[0]	O	Func3	O(H)
XM0CSN_5	SROM_CSn[5]	O	NFCSn[3]	O			ONANDXL_CSn[1]	O	Func3	O(H)

SROM_CSn[5]

2.2.25 PORT GROUP MP0_1 CONTROL REGISTER

There are six control registers, namely, MP0_1CON, MP0_1DAT, MP0_1PUD, MP0_1DRV, MP0_1CONPDN and MP0_1PUDPDN in the Port Group MP0_1 Control Registers.

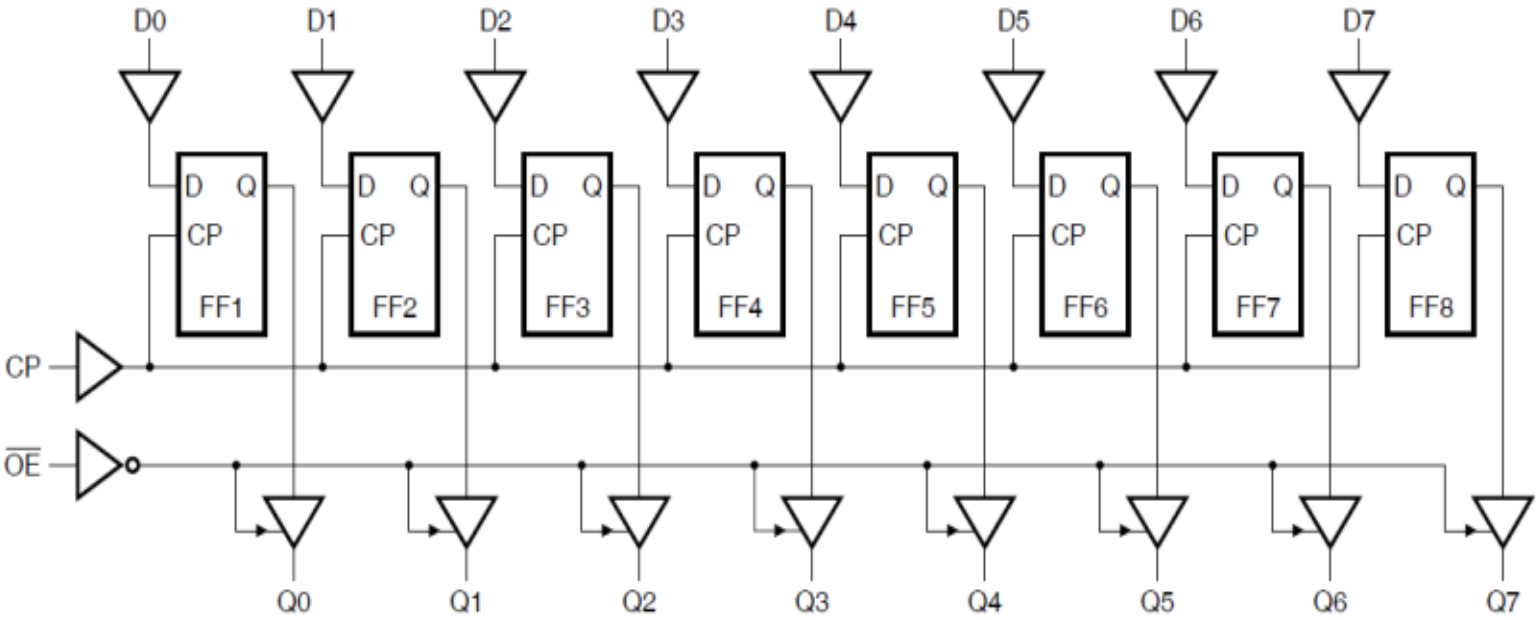
2.2.25.1 Port Group MP0_1 Control Register (MP0_1CON, R/W, Address = 0xE020_02E0)

MP0_1CON	Bit	Description	Initial State
MP0_1CON[7]	[31:28]	0000 = Input 0001 = Output 0010 = EBI_WEn 0011 ~ 1110 = Reserved 1111 = Reserved	0010
MP0_1CON[6]	[27:24]	0000 = Input 0001 = Output 0010 = EBI_OEn 0011 ~ 1110 = Reserved 1111 = Reserved	0010
MP0_1CON[5]	[23:20]	0000 = Input 0001 = Output 0010 = SROM_CSn[5] 0011 = NFCSn[3] 0100 = Reserved 0101 = ONANDXL_CSn[1] 0110 ~ 1110 = Reserved 1111 = Reserved	0101

(Ref) S5PV210 CPU User Manual (S5PV210_Usernamual_Rev1.0.pdf) p.23~24

74HC574

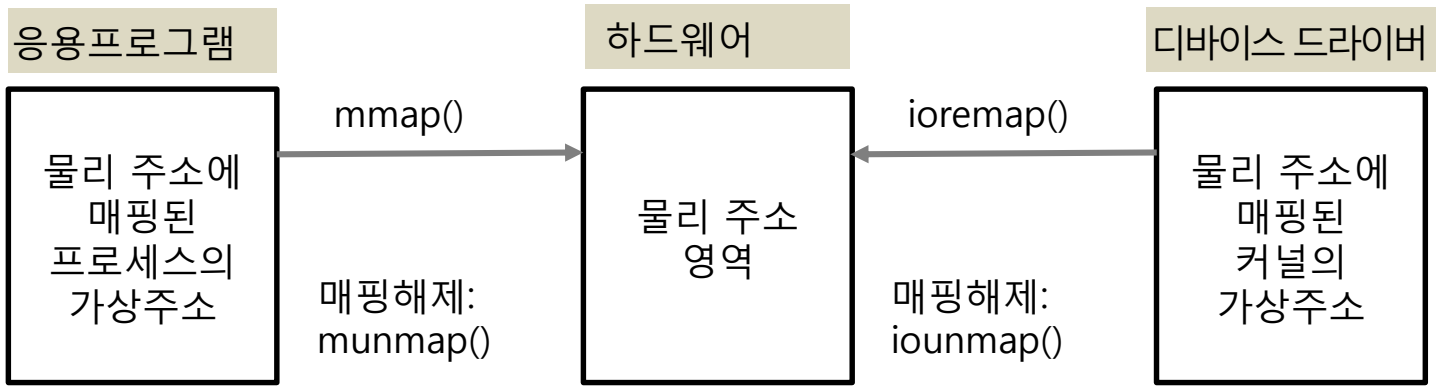
- 74HC574
  - 8개의 D flip-flop으로 구성



CPU와 PC100간의 AB/DB 연결

mmap vs ioremap

- mmap()과 ioremap()
  - 디바이스의 메모리 일부를 현재 프로세스의 메모리 영역으로 mapping 하는 방법
  - ioremap()
    - 해당 메모리의 물리주소를 커널의 가상주소에 매핑
  - mmap()
    - 응용 프로그램의 가상 주소에 해당 메모리의 물리주소를 매핑



mmap() vs ioremap()

- mmap()
  - 응용 프로그램에서 I/O장치의 physical address에 직접 접근하고자 할 때 사용
  - 즉 사용자 레벨에서 커널 레벨 (디바이스 접근)로 직접 접근
- ioremap()
  - 커널에서 I/O 장치의 physical address에 직접 접근하고자 할 때 사용
  - 커널 수준(Device driver 내)에서 직접 커널 수준(디바이스 접근)으로 접근
  - Device driver 내에서 작성시 드라이버 코드 내에 mmap대신 ioremap사용

- 코드를 작성할 때 ioremap을 사용해서 하는 것이 safe한 프로그램을 만드는 관점에서는 정석임
- Mmap의 장점은 응용 프로그램에서 직접 I/O 장치로 접근함으로 속도가 빠름.

ioremap 함수를 이용한 LED 제어

- 확장보드(PC100)에 있는 8개의 LED 제어
- 응용프로그램

  # ./ioremap_led_test 7
  - D7 LED의 불이 켜짐
  - Argument의 값이 '0'이면 8개 LED off, '9'이면 모두 ON
- 드라이버 작성 관련하여
  - LED의 물리주소는 0xA800_0000임
  - Major 번호는 246번으로 할당하는 것으로 함 (major.h에서 할당되지 않은 번호 선택함)
  - 드라이버 소스 파일명: ioremap_led_dd.c
  - 디바이스 노드 이름은 **iom_led**로 함

LED 구동 디바이스 드라이버 (ioremap_led_dd.c)

root@esp:~# **mkdir /root/work/dd/led_ioremap**

root@esp:~# **cd /root/work/dd/led_ioremap**

root@esp:~/work/dd/led_ioremap# **vi ioremap_led_dd.c**

```
/* LED Driver using ioremap function */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <asm/io.h> /* outb() */
#include <asm/uaccess.h> /* copy_from_user() */

#define DEV_NAME "iom_led"
#define IOM_LED_MAJOR_NUM 246
#define IOM_LED_ADDRESS 0xA8000000 /* physical addr for LED's */

MODULE_LICENSE("GPL");

static int ledport_usage = 0;
static unsigned char *iom_led_addr;

int init_iom_led(void);
void cleanup_iom_led(void);
module_init(init_iom_led); /* % insmod */
module_exit(cleanup_iom_led); /* % rmmod */
int iom_led_open (struct inode *, struct file *);
int iom_led_release (struct inode *, struct file *);
ssize_t iom_led_write (struct file *, const char *, size_t, loff_t *);
```

ioremap_led_device.c (2)

```
struct file_operations iom_led_fops = {
    .owner = THIS_MODULE,
    .open = iom_led_open,
    .release = iom_led_release,
    .write = iom_led_write,
};

int major_num = 0;

int iom_led_open (struct inode *inode, struct file *filp)
{
    if( ledport_usage != 0 )        return -EBUSY;
    ledport_usage = 1;
    return 0;
}

int iom_led_release (struct inode *inode, struct file *filp)
{
    ledport_usage = 0;
    return 0;
}

ssize_t iom_led_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos)
{
    unsigned char led_data;

    if (copy_from_user(&led_data, buf, count))       return -EFAULT;
    outb(led_data, (unsigned int)iom_led_addr);
    return count;
}
```

ioremap_led_device.c (3)

```
int __init init_iom_led(void)
{
   int major_num;
   major_num = register_chrdev(IOM_LED_MAJOR_NUM, DEV_NAME, &iom_led_fops);

   if ( major_num < 0 ) {
      printk(KERN_WARNING"%s: can't get or assign major number %d₩n", DEV_NAME, IOM_LED_MAJOR_NUM);
      return major_num;
   }

   iom_led_addr = ioremap(IOM_LED_ADDRESS, 0x1);

   printk("Success to load the device %s. Major number is %d₩n", DEV_NAME, IOM_LED_MAJOR_NUM);
   return 0;
}

void __exit cleanup_iom_led(void)
{
   iounmap(iom_led_addr);

   unregister_chrdev(IOM_LED_MAJOR_NUM, DEV_NAME);
   printk("Success to unload the device %s...₩n", DEV_NAME);
}
```

LED 구동 응용 프로그램 (ioremap_led_test.c)

root@esp:~/work/dd/led_ioremap# **vi ioremap_led_test.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define LED_DEV "/dev/iom_led"

unsigned char led_hex_data[] = {0x00, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f, 0xff};

int main(int argc, char *argv[])
{
   int fd;
   int led_index;

   if ( argc != 2 ) {
      printf("Usage: %s number (ex) %s 2 [0:9]..₩n", argv[0], argv[0]);
      return -1;
   }

   led_index = atoi(argv[1]);
   if (led_index < 0 || led_index > 9 ) {
      printf("Invalid Range!! (%d)₩n", led_index);
      return -1;
   }
```

ioremap_led_test.c (2)

```
 fd = open(LED_DEV, O_WRONLY);
   if ( fd < 0 ) {
      printf("Device open error (%s)...!!₩n", LED_DEV);
      return -1;
   }

   write (fd, &led_hex_data[led_index], sizeof(led_hex_data[led_index]));
   close (fd);
   return 0;
}
```

컴파일

% vi Makefile

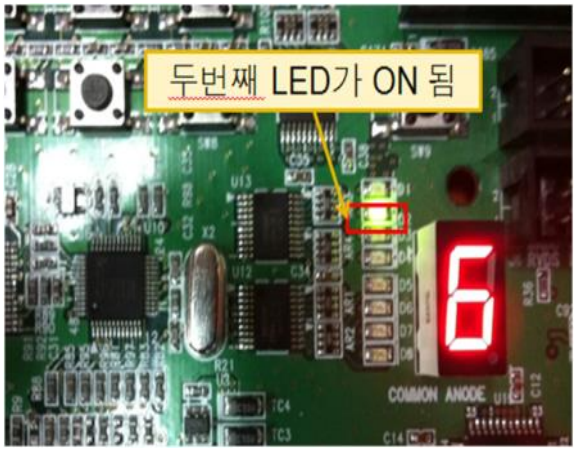
```
obj-m = ioremap_led_dd.o

CC = arm-linux-gcc
KDIR = /root/download/kernel-2.6.35

PWD = $(shell pwd)

TEST_TARGET = ioremap_led_test
TEST_SRCS = $(TEST_TARGET).c

all: module test_pgm

module:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
test_pgm:
	$(CC) $(TEST_SRCS) -o $(TEST_TARGET)
clean:
	rm -rf *.ko rm -rf *.o
	rm -rf *.symvers *.order
	rm -rf $(TEST_TARGET)
```

% make

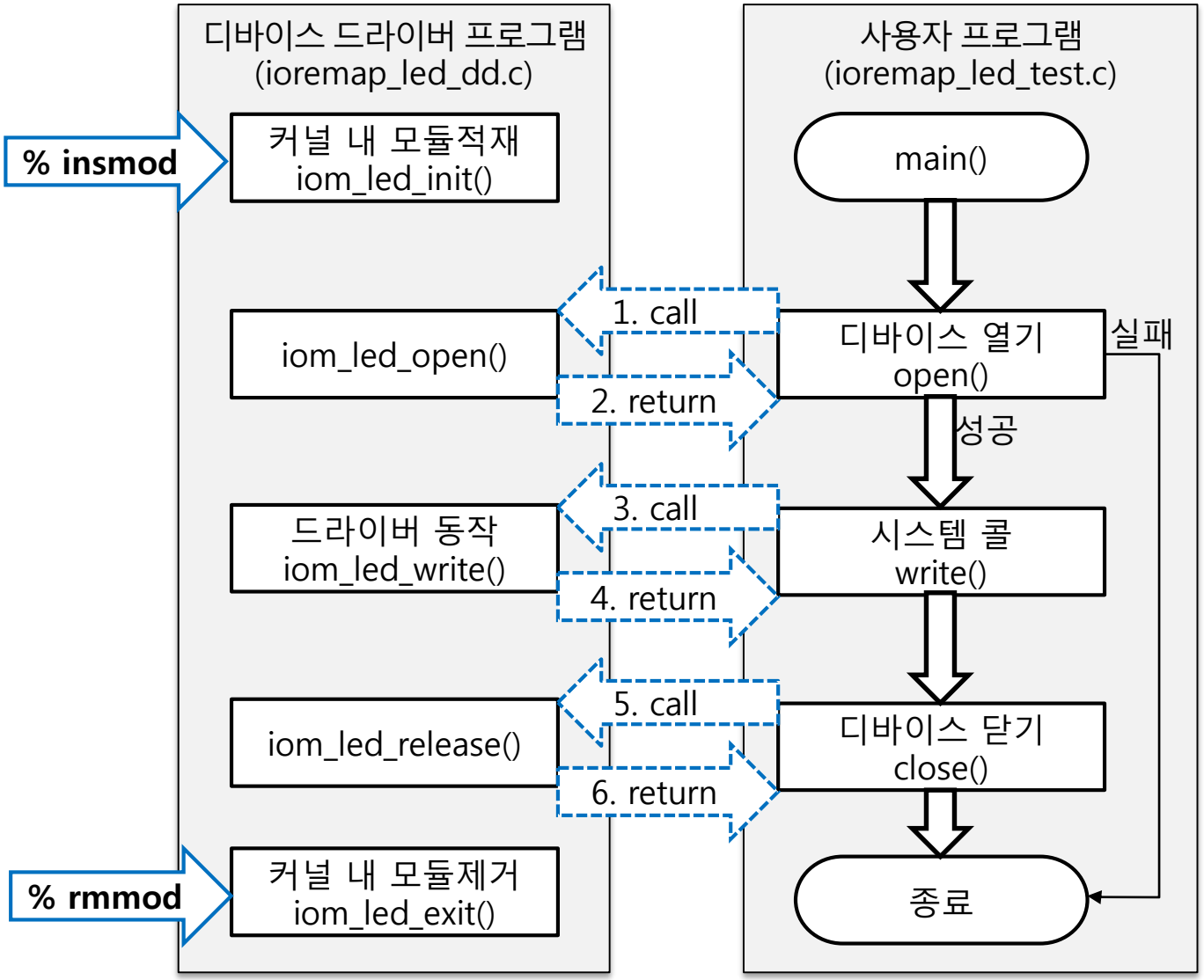
실행

- 타겟보드에서 실행
  - Host 컴퓨터의 /root 디렉토리를 NFS로 연결

```
# cd /root/nfs/work/dd/led_ioremap
# insmod ioremap_led_dd.ko
# mknod /dev/iom_led c 246 0
# ./ioremap_led_test 2
# ./ioremap_led_test 9
# ./ioremap_led_test 0
```



LED driver 구동 원리



mmap 함수를 이용한 LED 제어

- 확장보드(PC100)에 있는 8개의 LED를 1초 간격으로 번갈아가며 켜지도록 LED를 제어하는 프로그램을 작성. Ct기+C를 누르면 프로그램 종료됨
- LED 연결 분석
  - SROM bank5에 연결됨
  - CPU의 address bus Xm0ADDR[12:9]="0000"이어야 함
    - PC100 확장보드의 Xm0ADDR[20:17]이 CPU Xm0ADDR[12:9]에 연결됨
    - 따라서 LED에 접근하기 위한 주소는 0xA000_0000임

void *mmap()

led_addr = mmap(NULL, LED_MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, LED_PHY_ADDR);

- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
  - 메모리의 내용을 파일이나 디바이스에 대응하기 위해 사용하는 시스템 콜
  - void *addr, // 물리적 장치에 대해 메모리로 시작할 위치, 보통 0 (NULL)
  - size_t length, // 물리적 장치의 크기, 즉 확보될 메모리의 크기. Addr부터 length 만큼 매핑
  - int prot , // 읽기/쓰기와 같은 메모리의 특성 (헤더파일: <sys/mman.h>)
    - PROT_EXEC: 페이지에 실행될 수 있음
    - PROT_READ: 페이지는 읽기 가능
    - PROT_WRITE: 페이지는 쓰기 가능
    - PROT_NONE: 페이지를 접근할 수 없음
  - int flags, // 다른 프로세스와 공유할지 여부
    - MAP_FIXED
    - MAP_SHARED: 다른 프로세스와 공유하며, 사용하는 모든 프로세스는 동등한 권한을 가짐. 따라서 공유하는 프로세스로 인한 데이터 동기화가 필요하며, 이를 위해 msync(), munmap() 등이 사용됨
    - MAP_PRIVATE: 혼자만 사용할 경우. MAP_SHARED와 MAP_PRIVATE 둘 중 하나는 반드시 사용해야 함
  - int fd, // 물리적 장치의 디스크립터
  - off_t offset // 보통 0

mmap을 이용한 LED 구동 (led_mmap.c)

root@esp:~# **mkdir /root/work/dd/led_mmap**

root@esp:~# **cd /root/work/dd/led_mmap**

root@esp:~/work/dd/led_ioremap# **vi led_mmap.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/mman.h> /* PROT_WRITE */
#include <signal.h> /* SIGINT */

#define LED_MAP_SIZE    0x1000       /* 페이지 크기(4096)의 정수배 */
#define LED_PHY_ADDR    0xA8000000    /* led의 physical address */
#define LED       (*((volatile unsigned char *)(led_addr + 0)))

// LED를 표시하기 위한 값
// 0000(0) | 1110(1) | 1101(2) | 1011(3) ....
unsigned char val[] = { 0x00, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f};

void *led_addr;   // 메모리가 매핑될 포인터
int quit = 0;

void quit_signal(int sig) {  // Ctrl + C에 의해 중지되게 하기 위해 시그널을 사용한다.
    quit = 1;
}
```

led_mmap.c (2)

```
int main (int argc, char *argv[]) {
    int fd;
    static unsigned char led;

    // 주의: O_SYNC 플래그를 지정해야 캐시되지 않는다.
    fd = open( "/dev/mem", O_RDWR | O_SYNC );
    if (fd == -1) {
        perror("open(₩"/dev/mem₩")");
        exit(1);
    }
    // MAP_SIZE에 해당하는  페이지 만큼의 영역을 매핑한다.
    led_addr = mmap( NULL, LED_MAP_SIZE, PROT_WRITE, MAP_SHARED, fd, LED_PHY_ADDR );
    if (led_addr == MAP_FAILED) {
        perror("mmap()");
        exit(2);
    }
    signal(SIGINT, quit_signal );  // <ctrl+c> 를 누르면 종료되게 signal을 등록한다.
    printf("₩nPress <ctrl+c> to quit.₩n₩n");
    while(!quit) {
        LED = (val[led++ % 9]);
        sleep(1);
    }
    LED = 0xff;
    if (munmap(led_addr, LED_MAP_SIZE) == -1) { // 할당받았던 매핑 영역을 해제한다.
        perror("munmap()");
        exit(3);
    }
    close(fd);
    return 0;
}
```

컴파일 및 실행

[호스트]

% **arm-linux-gcc -o led_mmap led_mmap.c**

[Achro-210T]

cd ~/nfs/work/dd/led_mmap

./led_mmap