Embedded System Software [CSE4116]

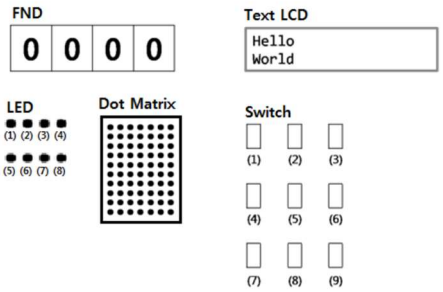
실습 2 주차 : Device Control Methods

Department of Computer Science and Engineering, Sogang University, Seoul, South Korea

Data-Intensive Computing and System Laboratory

1. FPGA 모듈

1.1. 구성 요소 및 사용방법



두 버튼 모두 ON으로 변경

2. 실습 1 주차 Remind

- 부팅 시, 바로 안드로이드 리눅스 저장 공간 확보 (On the target board-side, in the Minicom)
  dmseg가 출력되는 것을 무시하고 아래 명령어를 입력 (minicom에서 엔터 눌러보기)

```
$  mount -o rw,remount,size=6G /dev/mmcblk0p4 /data
```

  - df 명령어로 확인

```
df
Filesystem            Size      Used      Free   Blksize
/dev                922.0M    136.0K    921.8M   4096
/sys/fs/cgroup      922.0M     12.0K    922.0M   4096
/mnt/secure         922.0M      0.0K    922.0M   4096
/mnt/asec           922.0M      0.0K    922.0M   4096
/mnt/obb            922.0M      0.0K    922.0M   4096
/mnt/shm           1024.0K      0.0K   1024.0K   4096
/system             503.9M    456.4M     47.6M   4096
/data               128.0M     18.0M    110.0M   4096
root@achroimx:/ #
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT1
```

기존 (128M)

```
root@achroimx:/ # df
Filesystem            Size      Used      Free   Blksize
/dev                922.0M    136.0K    921.8M   4096
/sys/fs/cgroup      922.0M     12.0K    922.0M   4096
/mnt/secure         922.0M      0.0K    922.0M   4096
/mnt/asec           922.0M      0.0K    922.0M   4096
/mnt/obb            922.0M      0.0K    922.0M   4096
/mnt/shm           1024.0K      0.0K   1024.0K   4096
/system             503.9M    456.4M     47.6M   4096
/data                 6.0G    134.7M      5.9G   4096
root@achroimx:/ #
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT1
```

변경 (6GB)

- USB 디버깅(이 컴퓨터에서 항상 허용 체크) 허용해야 ADB push 작동
  (참고) [PROG] 버튼을 누르면 화면이 켜짐.

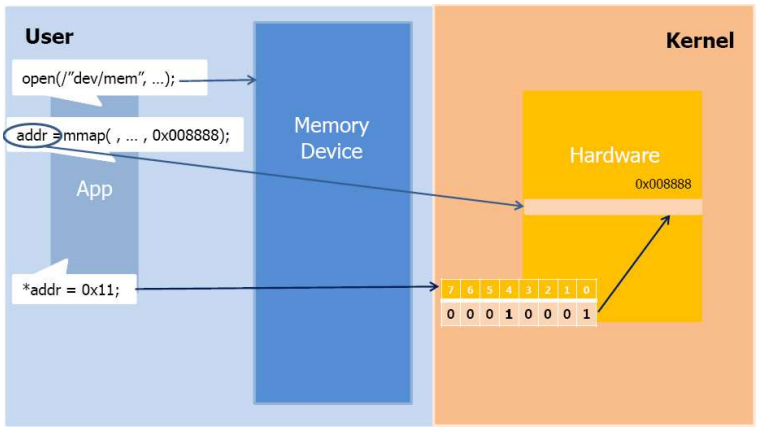
3. 실습 자료 다운 및 구성

```
$ cd ~/Downloads
$ wget --user=embe2024 --password=[칠판참고] ftp://dcclab.synology.me/embe2024/device_driver.tar
$ wget --user=embe2024 --password=[칠판참고] ftp://dcclab.synology.me/embe2024/device_test_modified_2.tar
```

- device_driver : Device 드라이버를 포함
- device_test_modified : Device 드라이버와 mmap 을 사용한 device 컨트롤 프로그램을 포함

4. Mmap()

- 응용 프로그램의 메모리 영역에 Hardware 물리 메모리 주소를 mapping 하는 방법
- Device Driver 를 사용하지 않음.
- User program 이 device 에 명령을 내리거나 데이터를 보내는 register 의 주소를 자신의 메모리 공간을 mmap 을 통해 mapping 한 뒤, 해당 공간에 데이터를 쓰거나 읽어서 device 를 직접 제어.
- 레지스터의 주소를 알아야 하고, 어떤 레지스터에 값을 썼을 때 device 가 어떻게 동작하는 지 등을 모두 알아야한다. (device driver 가 하던 일을 모두 user program 에서 수행)



- Memory device file (/dev/mem)을 open().
- mmap() 함수를 사용하여 물리 주소에 대한 포인터를 획득한다.
- 물리 주소에 대한 사용이 끝나면, munmap() system call 을 사용하여 mapping 을 해제한다.

4.1. 함수 설명

A. **void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)**

- [start+offset] ~ [start+offset+length] 만큼의 물리 메모리 공간을 mapping 할 것을 요청한다.
  - 단 [start+offset] 은 4KB 배수 단위의 주소여야 한다.
  - 일반적으로 start : NULL 또는 0 을 사용하고, mapping 되기 원하는 물리 메모리 주소를 offset 에 지정한다.
- prot : PROT_READ / PROT_WRITE / PROT_EXEC / PROT_NONE
- flags : MAP_SHARED / MAP_PRIVATE / MAP_FIXED
- fd : Device file 에 대한 file descriptor

B. **int munmap(void *addr, size_t length)**

- *addr 에 mapping 된 물리 메모리 주소를 해제한다.
- length : mapping 된 메모리의 크기 (mmap 에서 지정했던 length 와 같은 값을 준다).

4.2. Using LED Mmap()

- device_test 폴더에서 fpga_test_led_mmap.c 파일을 host 에서 cross-compile 하여 보드로 전송

```
$ arm-none-linux-gnueabi-gcc -static -o fpga_test_led_mmap_modified
  fpga_test_led_mmap_modified.c
$ adb push fpga_test_led_mmap_modified /data/local/tmp
```

  - (참고) tar -xvf 명령어를 사용하여 압축 해제

- 보드에서 실행(minicom)

```
$ ./fpga_test_led_modified [0-255]
```

- 코드 설명 참고자료(소스코드를 이해할 것)
  - LED data 레지스터 주소 : 0x08000016

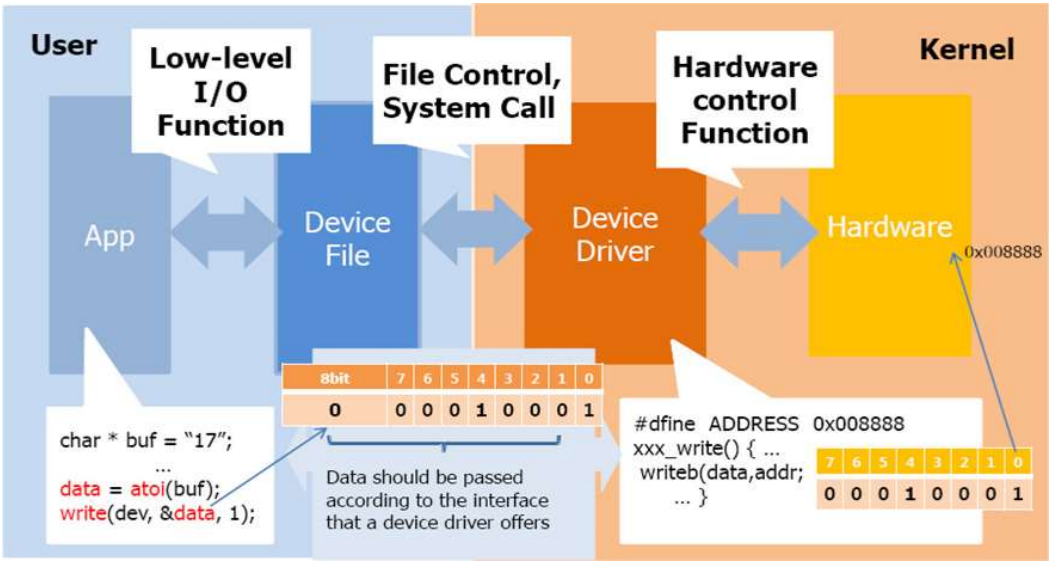
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#define FPGA_BASE_ADDRESS 0x08000000//fpga 모듈의 address로, LED의 물리주소를 지정하기 위한 base address
#define LED_ADDR 0x16             //0x08000016(0x08000000+16)은 LED의 data 레지스터 주소
//user가 입력한 숫자에 따라 led들을 켜는 프로그램
int main(int argc, char **argv)
{
    int fd,i;

    unsigned long *fpga_addr = 0;
    unsigned char *led_addr =0;
    unsigned char data;
    //argument 개수 맞는지 check
    if(argc!=2) {
        printf("please input the parameter! \n");
        printf("ex)./test_led [0-255]\n");
        return -1;
    }
    data=atoi(argv[1]);  //ex : digit 5 -> 0000 0101 (2진수로 바꿨을 때 1로 표현된 곳의 LED를 킴)

    if( (data<0) || (data>255) ){ //입력한 argument가 0~255, 즉 8bit로 표현가능한 범위인지 check
        printf("Invalid range!\n");
        exit(1);
    }

    fd = open("/dev/mem", O_RDWR | O_SYNC); //memory device open
    if (fd < 0) {  //open fail check
        perror("/dev/mem open error");
        exit(1);
    }
    //메모리 0x08000000 번지를 mmap (4KB mapping)
    fpga_addr = (unsigned long *)mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd, FPGA_BASE_ADDRESS);
    if (fpga_addr == MAP_FAILED) //mapping fail check
    {
        printf("mmap error!\n");
        close(fd);
        exit(1);
    }
    //led data 레지스터의 주소값 0x08000016(0x08000000+16)
    led_addr=(unsigned char*)((void*)fpga_addr+LED_ADDR);

    *led_addr=data; //led 레지스터에 data값 write(data값에 따라 led device의 켜지는 led 변경됨)

    sleep(1);
    //for read
    //해당 테스트 프로그램에서는 바로 직전 write했던 것을 다시 read하는 것이라 의미가 없지만
    //앞의 write 부분을 주석친다면, led 현재 상태를 read해옴
    data=0;
    data=*led_addr; //read led
    printf("Current LED VALUE : %d\n",data);

    munmap(led_addr, 4096); //mapping 해제
    close(fd);
    return 0;
}
```

5. Device Driver

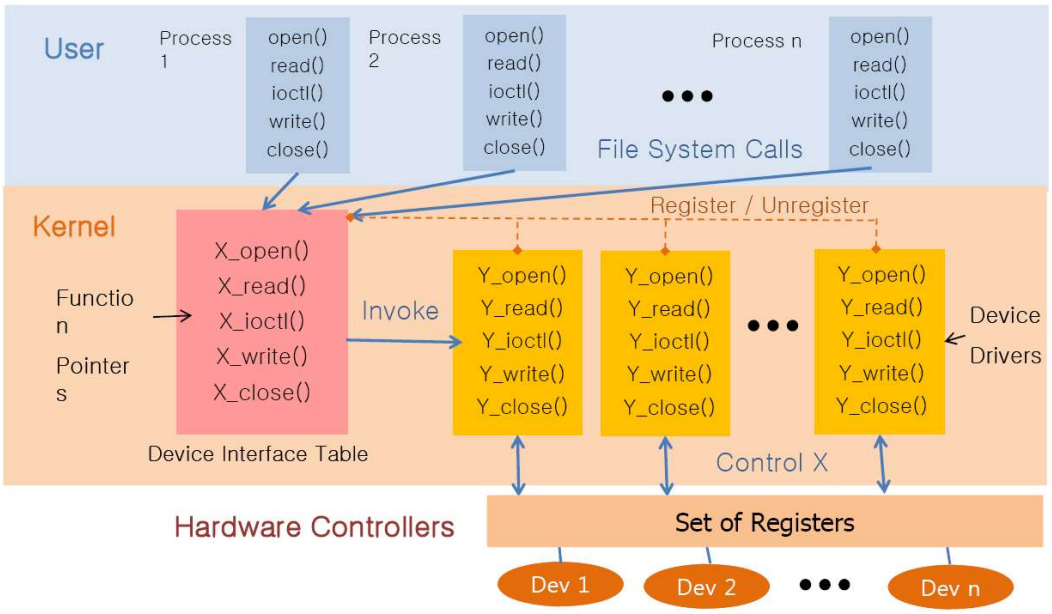
- User Program 이 Device 파일을 open 하여 read/write 와 같은 operation 을 통해 명령을 내리고, Device Drvier 가 이 명령을 해석해 device 를 제어함(값을 쓰거나 읽는 등)
- 따라서 mmap 과는 다르게 세부적인 mechanism(어떤 주소 값에 데이터를 쓰면 device 로 전송)등을 몰라도 사용 가능함.

5.1. Overview

A. Call Path



B. Device Driver Overview



C. **Low Level I/O Functions**

Low-Level I/O functions to control devices	Function
open()	Opens a file or device
close()	Closes a file or device
read()	Read from an open file decriptor
write()	Write to an open file descriptor
lseek()	Reposition read/write file offset
ioctl()	Performs a variety of device-speific control functions
fsync()	Synchronize with hardware

[추가 설명]

open()	int open(const char *pathname, int flags, *mode_t mode(optional)*); - path name : 경로 정보가 포함된 파일명 - flags : O_RDONLY, O_WRONLY, O_RDWR - mode : O_CREAT, O_APPEND, O_TRUNC - file descriptor 의 값 (fd) return
close()	int close(int fd) - return 0 : 성공적으로 닫힘 - return -1: 파일 닫기 실패 전역변수 errno에 오류번호 등록
read()	ssize_t read(int fd, void *buf, size_t count) - 실패 시 return -1 성공 시 읽어온 파일의 size 반환 - fd : 대상 파일 - buf : 파일을 읽어올 string의 pointer - count : 입력할 문자 수
write()	ssize_t write(int fd, const void *buf, size_t count) - 성공 시 write 한 작성 된 글자 수를 반환, 실패 시 return -1 - fd : 대상 파일 - buf : 쓸 내용이 입력되어 있는 pointer - count : 입력할 문자 수

5.2. Using Device Driver (On the Device-side, Minicom)

- Host 에서 다운받은 device_driver 폴더를 압축해제하고 디렉토리 전체를 보드로 전송하여
  보드 Minicom 에서 해당 디렉토리로 이동
- Device Driver 모듈 삽입

```
$ insmod [module]
```

```
[root@ACHRO module]# insmod fpga_led_driver.ko
[ 4464.076231] init module, fpga_led major number : 260
[root@ACHRO module]#
```

성공 시 major number 출력

- 삽입된 모듈 확인

```
$ lsmod
```

```
[root@ACHRO module]# lsmod
Module                  Size  Used by
fpga_dot_driver         1539  0
fpga_fnd_driver         1685  0
fpga_led_driver         1593  0
[root@ACHRO module]#
```

```
[root@ACHRO module]# cat /proc/modules
fpga_dot_driver 1539 0 - Live 0xbf018000
fpga_fnd_driver 1685 0 - Live 0xbf014000
fpga_led_driver 1593 0 - Live 0xbf010000
[root@ACHRO module]#
[root@ACHRO module]#
```

cat /proc/modules 로도 가능함

- (참고) 삽입된 모듈 제거

```
$ rmmod [module]
```

- Device 파일 생성

```
$ mknod /dev/[device name] [type] [major] [minor]
```

```
[root@ACHRO module]# mknod /dev/fpga_led c 260 0
[root@ACHRO module]# ls /dev/fpga*
/dev/fpga_led
```

ls -l /dev 명령어를 통해 device 파일 확인 가능

- (참고) 위 과정(Device Driver 모듈 삽입 & Device 파일 생성)은 이미 만들어진 스트립트 사용 가능

```
$ sh fpga_insmod.sh
```

- 로그 레벨 변경

```
$ echo "7 6 1 7" > /proc/sys/kernel/printk
```

로그 레벨을 변경하지 않을 시 printk 가 출력되지 않음
cat /proc/sys/kernel/printk 로 로그 레벨 확인 가능

A. (참고) Device 별 파일 이름과 major number

num	Device	Address	Node	Major
1	LED	0x0800_0016	/dev/fpga_led	260
2	Seven Segment (FND)	0x0800_0004	/dev/fpga_fnd	261
3	Dot Matrix	0x0800_0210	/dev/fpga_dot	262
4	Text LCD	0x0800_0090	/dev/fpga_text_lcd	263
5	Buzzer	0x0800_0070	/dev/fpga_buzzer	264
6	Push Switch	0x0800_0050	/dev/fpga_push_switch	265
7	Dip Switch	0x0800_0000	/dev/fpga_dip_switch	266
8	Step Motor	0x0800_000C	/dev/fpga_step_motor	267
EN	Demo Register	0x0800_0300	N/A	N/A

5.3. Test 프로그램 실행

Device Driver 별 실행 파일을 Cross-compile, 아래는 이미 만들어진 스크립트

```
$ sh execution.sh
```

이후 바이너리 파일을 보드로 전송

```
$ adb push *.out /data/local/tmp
```

보드에서 프로그램을 실행

A. **./fpga_test_dot.out [0-9]**

   Dot matrix 에 출력

B. **./fpga_test_fnd.out [maximum 4 digit integer]**

   Fnd 에 숫자 출력

C. **./fpga_test_led.out [0-255]**

   Led 불 들어오는 것 조정 (숫자를 2 진수로 표현했을 때 각 자리 값이 1 이면 불이 들어옴)

D. **./fpga_test_text_lcd.out [maximum 16 characters] [maximum 16 characters]**

   Text LCD 에 string 1 을 첫번째 줄에, string 2 는 두번째 줄에 출력

E. **./fpga_test_push_switch.out**

   어떤 스위치가 눌렸는지 실시간 print

F. **./readkey.out**

   Back,prog,vol+,vol- 버튼 눌리고, 땔 때 print

G. **./fpga_test_buzzer.out**

   소리 키고 끄고 반복

H. **./fpga_test_dip_switch.out**

   Reset 버튼(fpga 모듈) push 여부 실시간 print

I. **./fpga_test_step_motor.out [] [] []**

   Motor 돌리는 프로그램

J. **./fpga_test_push_switch.out**

   Switch 를 실행하는 프로그램, 스위치가 눌리면 0 출력

6. 실습 과제

다음 항목이 모두 포함되게 사진을 찍어 사이버 캠퍼스에 업로드

- [제출형식] : 학번_이름.jpg
- 형식 틀릴 시 감점 1 점

- LED 8 번(D8) ON
- Dot 프로그램으로 자신의 학번 맨 뒷자리를 출력
- FND 프로그램으로 자신의 학번 뒤에서 4 자리를 출력
- LCD 프로그램으로 첫번째 줄에는 오늘 날짜, 두번째 줄에는 자신의 학번 모두를 출력

[예시]