Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

RC-CAR let's get it

I2C 커널 드라이빙

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

목차

1 디바이스드라이버란?

: 모니터 키보드 마우스 모두 파일이며, 파일시스템을 통해 디스크에 저장되어 있는 파일도 파일이다.
→ 장치 파일 → 정규파일

"모든것은 파일이다" 이므로 사용자 태스크는 현재 접근하려는 파일이 '모니터'인지 '키보드'인지 '정규파일'인지 신경 쓰지 않고, 그저 파일에만 접근하면 된다.
→ open(), read(), write(), close() 등의 일관된 함수 인터페이스를 통해 정규파일, 디바이스파일 에 접근하는 것이 가능하다.

: 태스크가 open한 파일과 연관되어 있는 정보를 관리하기 위해 사용하는 구조체
- 2개 이상의 태스크가 하나의 파일에 접근할 때, 각각의 offset등과 같은 정보를 관리해야함
- 태스크 관련 정보를 유지하는 용도로 파일 객체를 사용
- 각 태스크가 아이노드 객체에 접근하는 동안만 메모리상에 유지

1. 사용자 태스크 관점에서의 디바이스 드라이버

: 리눅스에서는 장치들을 파일화하여 관리한다. 이러한 파일들을 장치파일 이라 부르고, /dev디렉토리에 위치하게 된다.
사용자 수준 응용 프로그램에서 접근하는 **장치 파일**이라는 것은 VFS가 제공하는 **'파일객체'**이다. 파일 객체에 사용자 태스크가 행할 수 있는 연산은 "linux/include/linux/fs.h" 파일 내에 struct file_operations 으로 정의 되어 있다. 사용자 태스크는 이 operations 객체내에 open(), read(), write(), release() 등의 여러 함수들을 이용하여 파일 객체 즉, 장치 파일에 접근할 수 있다.

(디바이스 파일에서 접근하는 방법이고, 정규파일에서 접근한다면 파일 시스템에서 제공하는 함수를 호출한다 – 커널 책 5장을 살펴보자)

2. 개발자 태스크 관점에서의 디바이스 드라이버 (장치파일과 디바이스 드라이버의 관계)

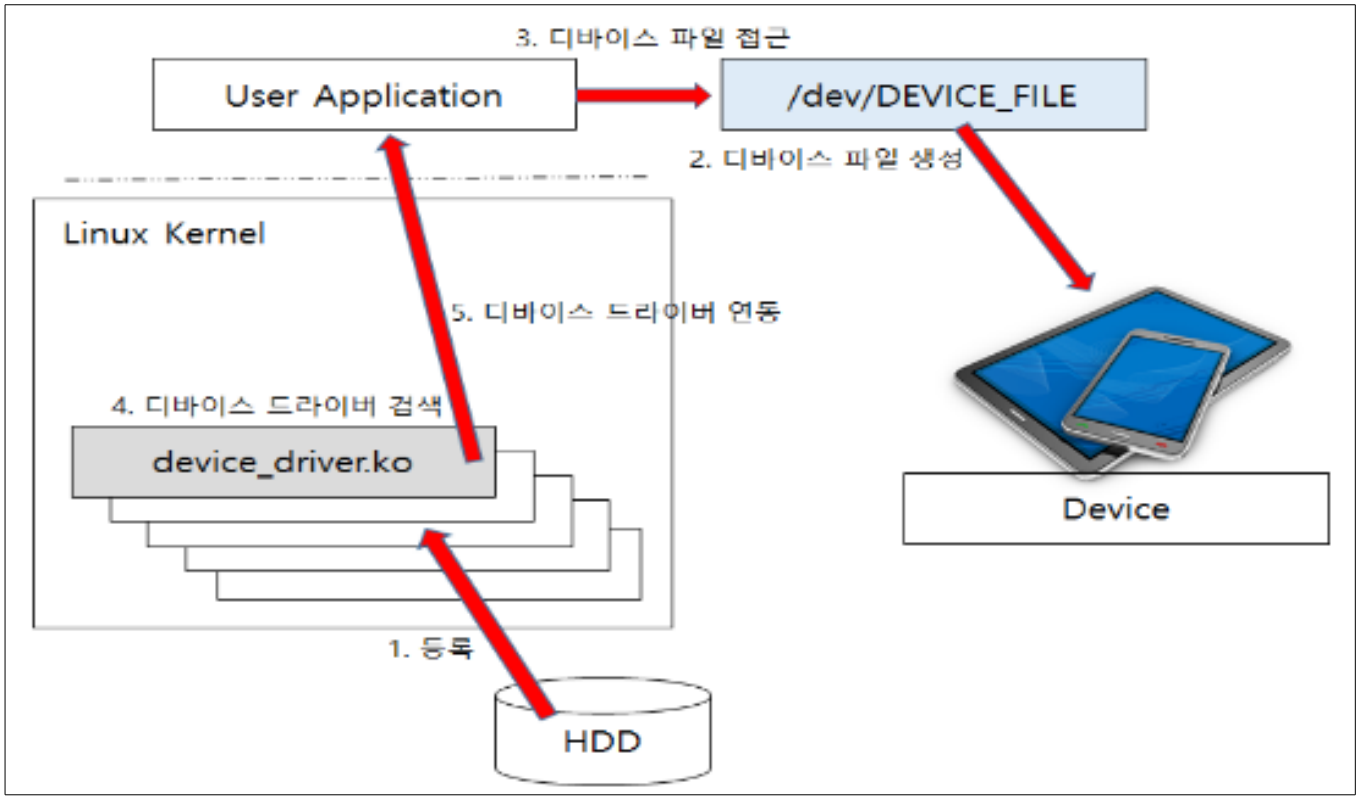
유저 영역에서 장치 파일에 접근하는 것은 file_operations 구조체 안의 함수들을 가지고 접근을 한다고 했다. 그렇다면 접근한다는 장치 파일은 실제 하드웨어를 뜻할까?? 장치 파일에는 실제 장치에 대한 정보가 아니라 작동 시킬 디바이스 드라이버에 대한 정보가 들어있다. 1. 디바이스 드라이버 타입정보 2. 주번호 3. 부번호 라는 세가지 정보가 이 장치 파일 안에 들어있다. 유저 영역에서 장치 파일에 접근하여 파일을 열었을 때, 리눅스 커널은 장치 파일에 들어있는 정보를 가지고 개발자가 실제 장치에 접근 하기 위해 미리 개발하고, 커널에 등록 시킨 디바이스 드라이버를 찾아 실행시킨다. 여기서 개발자 관점에서 작성하는 디바이스 드라이버는 실제 장치에 접근 하기 위해 호출할 함수를 정의하고 구현하는 것을 말한다.

2 드라이버 구동 과정 분석 – 1

1. 디바이스 드라이버 등록

가장 먼저 수행해야 할 사항은 디바이스를 제어할 디바이스 드라이버를 리눅스 커널에 등록 시키는 것이다. 실제 디바이스를 제어하는 것은 리눅스 커널이 아니라 커널 내에 등록되어 있는 디바이스 드라이버이기 때문이다.
등록시키는 방법은 Kernel Module 형태로 컴파일한 뒤 *.ko 파일을 “insmod” 명령어를 통해 커널 메모리 영역 내에 추가하는 형태로 등록시키거나, 아예 커널 소스 내에 포함시켜 같이 컴파일 시키는 방법이 있다.

주로 개발 초기에 커널 모듈 형태로 개발을 하고, 개발이 종료되면 커널 소스 내에 포함 시켜 컴파일 하거나, 혹은 kenel module 파일을 시스템이 부팅할 때 자동적으로 삽입되도록 “insmod”명령어 실행 부분을 미리 시스템 부팅 스크립트 파일들(ex/etc/inittab)에 가입해 놓는 방식으로 개발이 된다.



2. 디바이스 파일 생성

리눅스는 자신이 관리해야할 모든 자원들을 “파일” 단위로 관리한다. 일반 파일들은 물론 실행되고 있는 프로세스들도 모두 파일 단위로 관리한다. 마찬가지로, 주변 디바이스들 역시 파일로 관리하고 있으며, 이 때 사용되는 파일을 “디바이스 파일”이라고 한다.

디바이스 파일은 실제 디바이스를 가리키는 c언어에서의 포인터와 같은 역할을한다. User Application이 직접 주변 디바이스를 제어하거나 접근할 수 없기 때문에, 간접적으로 디바이스 파일을 통해 해결할 수 밖에 없다. 즉, User Application이 디바이스에 무언가 데이터를 쓰고 싶다면, 디바이스와 연결된 디바이스 파일에 쓰면 된다. 이러한 디바이스 파일은 “mknod” 명령어를 통해 생성할 수 있다.

3. 디바이스 파일 접근

User Application은 가상 메모리 기법에 의해 직접적으로 디바이스에 접근할 수 없다. 따라서 실제 디바이스에 접근하여 User Application이 원하는 대로 제어하는 것을 Device Driver가 담당하게 된다. 이러한 Device Driver는 독립적으로 실행될 수 가 없다. 오직 접근하려고 할 때 리눅스 커널이 이를 알아채고 실행시켜주어야만 한다.

알아채는 방식은 주변 디바이스와 일대일로 연결(링크)되어 디바이스 파일을 User Application이 open() 함수를 이용하여 열게 되면, open() 함수를 서비스해주기 위해 리눅스 커널이 해당 파일로 접근하게 되고, 이 파일이 디바이스 파일임을 알게 되면 해당 디바이스를 제어하겠다고 자신에게 등록한 Device Driver들을 검색하게 된다. 만약 해당 Device Driver를 찾지 못하게 되면, 서비스를 요청한 User Application에 오류 메세지를 보내준 후 프로그램을 종료 시키고, 해당 Device Driver를 찾게 되면 실행시켜 연동되도록 서비스 해준다.

4. 디바이스 드라이버 검색

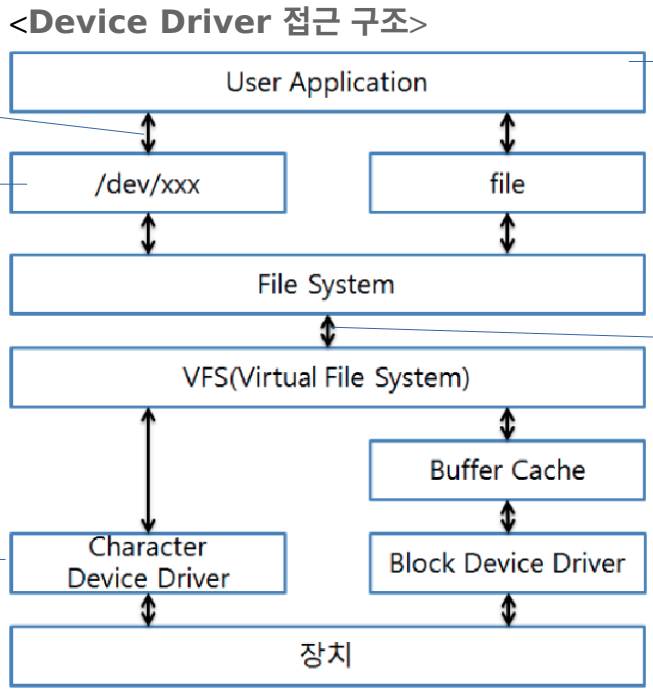
User Application이 특정 장치 파일을 열게 되면, 해당 장치 파일과 연결되어 있는 디바이스를 제어하겠다고 자신(Linux Kernel)에게 미리 등록한 Device Driver 가 있는지, 각 디바이스(Device Character, Block, Network) 그룹 별로 검색하게 된다.

5. 디바이스 드라이버 연동

검색이 되었다면 이를 실행시켜 User Application과 연동되게 해준다. 이후부터 User Application이 해당 디바이스 파일에 파일 제어 함수들을 이용하여 제어 요청을 하게 되면, Device Driver가 해당 요청을 받아 직접 디바이스를 제어하게 된다.

2 드라이버 구동 과정 분석 – 2

file_operations 객체내 함수들로 장치파일에 접근



유저영역
: 커널에게 디바이스 제어 요청

장치파일
: 디바이스 드라이버 타입정보
: 주번호, 부번호

유저에서 호출하는 함수의 인자에 담겨있는 파일이름을 보고, 해당 파일 시스템이 어떤 것인지 판단하고, 사용자가 원하는 일을 해 줄 수 있는 파일 시스템 고유의 함수를 호출

디바이스 드라이버
:호출할 함수 정의 및 구현

→ 파일 시스템을 통해 접근되는 **블록 디바이스 드라이버**, 사용자 수준 응용 프로그램이 장치파일을 통해 직접 접근하는 **문자 디바이스 드라이버**

리눅스 **장치 파일**은 /dev 디렉터리 밑에 존재하고, 이 장치 파일에는 **file_operation** 내의 함수들로 접근한다. 유저영역에서 장치 파일에 접근을 한 후, 실제 장치를 제어 하기 위해 호출할 함수를 정의하고 구현하는 것이 **디바이스 드라이버의 역할**이다. 즉 사용자 태스크는 디바이스 드라이버 개발자가 작성한 여러가지 함수를 모두 알 필요 없이 파일 객체에 정의된 함수를 호출하는 것 만으로 장치에 접근이 가능 한 것이다.

3 Project 에서의 디바이스 드라이버

Lidar와 스텝모터는 "/dev/i2c-0" 을 can통신은 “/dev/spi-0” 장치파일을 사용한다. 이미 i2c 와 spi 장치 드라이버는 xilinx 사에서 배포하는 petalinux 커널 안에 존재한다.

경로를 살펴보면 “~/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/xlnx-4.0/source/drivers/i2c/i2c-dev.c” 에 디바이스 드라이버가 있는 것을 찾아볼 수 있었다.

그렇다면 장치파일은 어디에 있을까..?

```
struct inode {
    umode_t         i_mode;
    unsigned short      i_opflags;
    kuid_t          i_uid;
    kgid_t          i_gid;
    unsigned int        i_flags;

#ifdef CONFIG_FS_POSIX_ACL
    struct posix_acl    *i_acl;
    struct posix_acl    *i_default_acl;
#endif

    const struct inode_operations   *i_op;
    struct super_block  *i_sb;
    struct address_space    *i_mapping;

#ifdef CONFIG_SECURITY
    void            *i_security;
#endif

    /* Stat data, not accessed from path walking */
    unsigned long       i_ino;
    /*
     * Filesystems may only read i_nlink directly.  They shall use the
     * following functions for modification:
     *
     *    (set|clear|inc|drop)_nlink
     *    inode_(inc|dec)_link_count
     */
    union {
        const unsigned int i_nlink;
        unsigned int __i_nlink;
    };
    dev_t           i_rdev;
    loff_t          i_size;
    struct timespec     i_atime;
    struct timespec     i_mtime;
    struct timespec     i_ctime;
    spinlock_t      i_lock; /* i_blocks, i_bytes, maybe i_size */
    unsigned short          i_bytes;
    unsigned int        i_blkbits;
    blkcnt_t        i_blocks;

#ifdef __NEED_I_SIZE_ORDERED
```

$ vi tags (path: tags 생성했던 폴더) 로 tags 파일을 연 상태에서 :tj file_operations

```
  # pri kind tag             파일
  1 F   s    file_operations   link-to-kernel-build/source/include/linux/fs.h
               struct file_operations {
  2 F   s    file_operations   xlnx-4.0/source/include/linux/fs.h
               struct file_operations {
:tj file_operations
```

```
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 (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
    int (*iterate) (struct file *, struct dir_context *);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    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 (*mremap)(struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, 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 (*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 (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
    int (*setlease)(struct file *, long, struct file_lock **, void **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
              loff_t len);
    void (*show_fdinfo)(struct seq_file *m, struct file *f);
#ifndef CONFIG_MMU
    unsigned (*mmap_capabilities)(struct file *);
#endif
};
```

```
blktrace_api.h      elf.h               i2c.h                ip_vs.h          media.h              netlink_diag.h
bpf.h               elfcore.h           i2o-dev.h            ipc.h            mei.h                netrom.h
lhs@lhs-Lenovo-YOGA-720-13IKB:~/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/xlnx-4.0/source/include/uapi/linux$
```

/sys/dev/ block, char

```
lhs@lhs-Lenovo-YOGA-720-13IKB:/sys/dev$ ls
block  char
lhs@lhs-Lenovo-YOGA-720-13IKB:/sys/dev$ cd char/
lhs@lhs-Lenovo-YOGA-720-13IKB:/sys/dev/char$ ls
108:0   10:228  10:55  10:63   116:5  13:35  13:70  13:78    189:3  1:8    242:0    254:0  4:14  4:21  4:29  4:36  4:43  4:50  4:58  4:65  4:72  4:8   4:87  4:94  7:128  7:3   89:3
10:1    10:229  10:56  116:1   116:6  13:63  13:71  13:79    189:4  1:9    242:1    29:0   4:15  4:22  4:3   4:37  4:44  4:51  4:59  4:66  4:73  4:80  4:88  4:95  7:129  7:4   89:4
10:183  10:231  10:57  116:10  116:7  13:64  13:72  13:80    1:1    226:0  243:0    4:0    4:16  4:23  4:30  4:38  4:45  4:52  4:6   4:67  4:74  4:81  4:89  5:0   7:130  7:5   89:5
10:184  10:234  10:58  116:11  116:8  13:65  13:73  13:81    1:11   226:128 244:0    4:1    4:17  4:24  4:31  4:39  4:46  4:53  4:60  4:68  4:75  4:82  4:9   5:1   7:131  7:6   89:6
10:200  10:235  10:59  116:2   116:9  13:66  13:74  189:0    1:3    239:0  244:1    4:10   4:18  4:25  4:32  4:4   4:47  4:54  4:61  4:69  4:76  4:83  4:90  5:2   7:132  81:0  89:7
10:223  10:236  10:60  116:3   13:32  13:67  13:75  189:128  1:4    239:1  244:2    4:11   4:19  4:26  4:33  4:40  4:48  4:55  4:62  4:7   4:77  4:84  4:91  5:3   7:133  89:0
10:224  10:237  10:61  116:33  13:33  13:68  13:76  189:2    1:5    240:0  249:0    4:12   4:2   4:27  4:34  4:41  4:49  4:56  4:63  4:70  4:78  4:85  4:92  7:0   7:134  89:1
10:227  10:54   10:62  116:4   13:34  13:69  13:77  189:25   1:7    241:0  253:65536 4:13  4:20  4:28  4:35  4:42  4:5   4:57  4:64  4:71  4:79  4:86  4:93  7:1   7:2    89:2
```

*** ctags 사용법**

Linux/include/linux/fs.h file_operation

우리가 Lidar 프로젝트를 만들었을 때, ctags를 사용하여 file_peration을 찾아보려고 한다.
우선 ctags 설정을 해줘보자. 리눅스 프로그래밍 수업 때 했었는데, 가물가물 할테니 다시 해보도록 한다.

우선 vi ~/.vimrc 와 vi ~/mkcscope.sh 를 열어 다음코드를 복붙해주자.

vi ~/.vimrc

```
"ctags 설정"
set tags=/root/compiler/gcc-4.5.0/tags
(이후에 set tags=/home/lhs/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/tags 로바꿔준다)
if version >= 500
func! Sts()
    let st = expand("<cword>")
    exe "sts ".st
endfunc
nmap ,st :call Sts()<cr>
func! Tj()
    let st = expand("<cword>")
    exe "tj ".st
endfunc
nmap ,tj :call TJ()<cr>
endif

"cscope 설정"
set csprg=/usr/bin/cscope
set nocsverb
cs add /root/compiler/gcc-4.5.0/cscope.out
(이후에 cs add /home/lhs/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/cscope.out 로바꿔준다)
set csto=0
set cst
func! Css()
    let css = expand("<cword>")
    new
    exe "cs find s ".css
    if getline(1) == ""
        exe "q!"
    endif
endfunc
nmap ,css :call Css()<cr>

func! Csd()
    let csd = expand("<cword>")
    new
    exe "cs find d ".csd
    if getline(1) == ""
        exe "q!"
    endif
endfunc
nmap ,csd :call Csd()<cr>

func! Csg()
    let csg = expand("<cword>")
    new
```

2. 명령어: vi ~/mkcscope.sh

```
#!/bin/sh
rm -rf cscope.files cscope.files
find . \( -name '*.c' -o -name '*.cpp' -o -name '*.cc' -o -name '*.h' -o -name '*.S' \) -print >
cscope.files
cscope -i cscope.files
```

```
    exe "cs find g ".csg
    if getline(1) == ""
        exe "q!"
    endif
endfunc
nmap ,csg :call Csg()<cr>
```

자이제 ctags 와 cscope를 다운로드하고, 내가 ctags를 사용하고자 하는 폴더로 들어가서 다음 명령어를 입력한다. 필자의 경우 Lidar 프로젝트에서 사용된 petalinux 커널을 분석할 것이므로 ~/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel 경로에 tags파일을 생성했다.

명령어: sudo apt-get install ctags cscope
명령어 : ctags -R
폴더로 이동했으면 ctags -R 을 입력하면 tags 파일이 생성됩니다

tags파일을 생성했으면 다음 과정을 진행 하자.

cd ~/ (홈으로)
명령어: chmod 755 ~/mkcscope.sh → mkcscope.sh 가 초록색이 되야함.
명령어: sudo cp ~/mkcscope.sh /usr/local/bin/
vi ~/.vimrc 빨간 글씨를 파란 글씨로 바꿔준다. 이때 tags 를 생성했던 파일로 경로설정을 제대로 해줘야 ctags를 사용할수 있다.

자 이제 설정이 끝났다. VFS가 제공하는 파일 객체에 명령을 내려주는 함수들이 있는 “linux/include/linux/fs.h” 안의 file_operations 함수를 찾아보자. $ vi -t file_operations

vi -t 와 다른방법으로는 우리가 vimrc에 ctags 설정을 해주었으므로 vi 환경 어느 곳에서나 ctags를 사용할 수 있다. 가령 내가 코드를 보고 있다가

이 함수에 대해 **:tj file_operations** or 해당 함수나 변수 이름 위에서 **Ctrl +]** 로 해당 함수를 찾을 수 있다.

뒤로가기는 **: po** 또는 **Ctrl + t**

디렉터리 찾기: find -name 'dev*' -type d → dev로 시작하는 디렉터리 찾음.

파일명으로 찾기 : find -name 'AAA*' → AAA로 시작하는 파일 찾음.

Find / 앞에 (v / v)를 붙이면 드라이브안 모든 파일에서 찾는것. (v는 띄워쓰기)

샘 새로운 mpu6050 코드
~/workspace_v8/mpu6050/hw_sw_co_design/petalinux/i2c_mpu6050/components/apps/i2c_mpu6050

라이다 프로젝트 캐릭터 디바이스 폴더 위치
~/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/xlnx-4.0/include/config/chr/dev

라이다 프로젝트 i2c 디바이스 드라이버 위치
~/i2c_proj/PETALINUX/i2c_lidar/build/linux/kernel/xlnx-4.0/source/drivers/i2c

라이다 프로젝트 i2c 유저단
~/i2c_proj/PETALINUX/i2c_lidar/components/apps/i2c_lidar