GPIO 디바이스 드라이버

목표

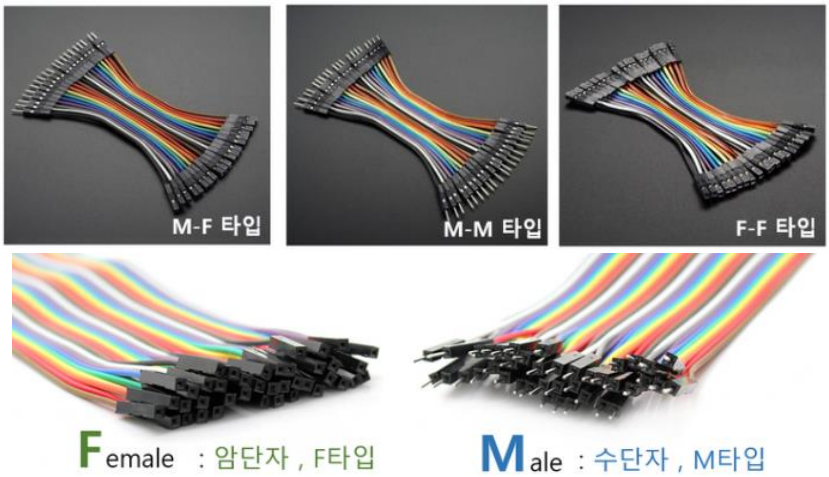
- Raspberry Pi에 자신이 제작한 GPIO 디바이스 드라이버를 커널에 빌드
- 자신이 제작한 GPIO 디바이스 드라이버를 직접 사용해보기

준비물

- 호스트 시스템에서 타겟 시스템의 Kernel 소스코드(4.19.x 이상 버전 권장)
  *호스트 시스템에 타겟 시스템의 Kernel 소스코드가 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조
- 호스트 시스템에서 타겟 시스템의 툴 체인 환경(Cross Compiler, Make 등)
  *호스트 시스템에 타겟 시스템의 툴 체인 환경이 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조
- 발광 다이오드(LED) 1개

- 슬라이드 스위치 1개

- 점퍼선 M-F
- 점퍼선 M-M

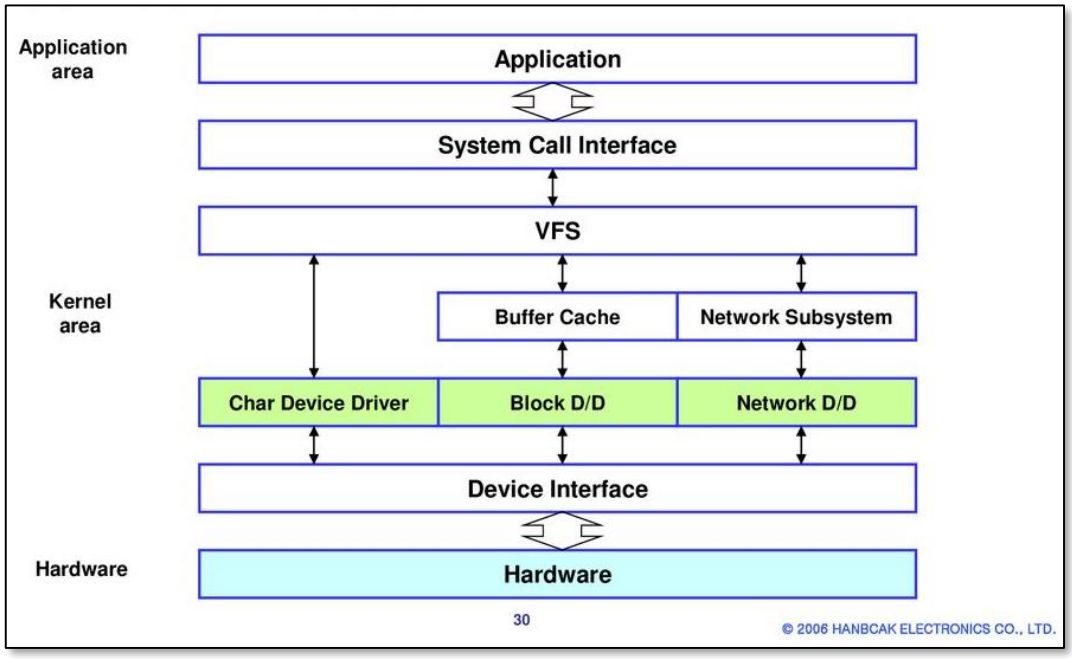


GPIO 디바이스 드라이버란?

- 리눅스는 디바이스(HDD, 키보드, 마우스 등)들을 파일로 취급하며 해당하는 디바이스 파일에 맞춘 드라이버를 제작하여 특정 디바이스를 제어한다.

- 본 실습에서 구현할 GPIO 디바이스 드라이버는 캐릭터 디바이스 드라이버 형태로 제작된다. 즉 버퍼나 캐시를 거치지 않고 유저 프로세스로부터 직접 데이터를 읽고 쓰는 디바이스 드라이버이다.

- 리눅스 디바이스 드라이버 구조



Raspberry Pi CPU 칩셋 데이터시트 살펴보기

- 본 실습에서 사용된 **Raspberry Pi 보드 버전은 4B**이다.

Raspberry Pi 칩셋 데이터시트는 Raspberry Pi 버전에 따라 상이하다.
Raspberry Pi 4B에는 BCM2711라는 CPU 칩셋이 사용된다.

- Raspberry Pi CPU 칩셋 살펴보기

칩셋 명	Raspberry Pi 버전
BCM2835	• Raspberry Pi 1A

	• Raspberry Pi 1A+ • Raspberry Pi 1B • Raspberry Pi 1B+ • Raspberry Pi Zero • Raspberry Pi Zero W
BCM2836 (아키텍처는 BCM2835와 동일함)	• Raspberry Pi 2B (구)
BCM2837 (아키텍처는 BCM2835와 동일함)	• Raspberry Pi 2B (신) • Raspberry Pi 3B • Raspberry Pi Compute Module 3
BCM2387B0 (아키텍처는 BCM2835와 동일함)	• Raspberry Pi 3A+ • Raspberry Pi 3B+ • Raspberry Pi Compute Module 3+
BCM2711	• Raspberry Pi 4B • Raspberry Pi 400 • Raspberry Pi Compute Module 4

- 이번 섹션에서는 Raspberry Pi 4B의 BCM2711 CPU 칩셋의 GPIO를 관리하는 레지스터 부분을 데이터시트를 통해 살펴본다.

- BCM2711 칩의 데이터시트는 아래 링크를 통해 접근할 수 있다.
  >> BCM2711 Datasheet Link : https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf

- GPIO를 관리하는 BCM2711 칩셋의 레지스터 Base 주소는 **0x7e200000**이다.

> **5.2. Register View**
>
> The GPIO has the following registers. All accesses are assumed to be 32-bit. The GPIO register base address is 0x7e200000.

데이터시트의 일부 내용 발췌

Base 주소는 RAM의 물리주소이다. 이 Base 주소가 CPU 레지스터 주소와 매핑 된다.
즉 Base 주소에 위치한 메모리 내용을 변경하게 되면 CPU에 위치한 GPIO 레지스터 값이 변경되는 것이다.

- GPIO 레지스터는 여러 개가 있으며 각 각의 역할이 존재한다.

Offset	Name	Description
0x00	GPFSEL0	GPIO Function Select 0
0x04	GPFSEL1	GPIO Function Select 1
0x08	GPFSEL2	GPIO Function Select 2
0x0c	GPFSEL3	GPIO Function Select 3
0x10	GPFSEL4	GPIO Function Select 4
0x14	GPFSEL5	GPIO Function Select 5
0x1c	GPSET0	GPIO Pin Output Set 0
0x20	GPSET1	GPIO Pin Output Set 1

>> BCM2711 칩셋에는 총 30개의 GPIO 레지스터가 존재한다. (BCM2711 데이터시트의 66Page 참조)

- 본 실습에서는 30개의 GPIO 레지스터 중 4개의 GPIO 레지스터를 사용할 것이다.
  사용되는 레지스터 이름과 역할은 다음과 같다.

레지스터 이름	역할
GPFSEL	• GPIO 핀 모드(Input/Output) 설정을 위해 사용
GPSET	• Output 모드로 설정된 GPIO 핀의 1(High) 출력을 위해 사용
GPCLR	• Output 모드로 설정된 GPIO 핀의 0(Low) 출력을 위해 사용
GPLEV	• Input 모드로 설정된 GPIO 핀에 입력되는 값을 읽기 위해 사용

- 본 실습에서는 GPIO 18번 핀을 통해 LED를 끄고 킬 것이며, GPIO 19번 핀을 통해 스위치로 들어오는 값을 읽어볼 것이다.

  GPIO 18번과 19번 핀의 모드 설정은 GPFSEL0~5 중 **GPFSEL1**을 통해 설정 가능하다.
  또한 GPIO 18번의 출력은 GPSET0~1과 GPCLR0~1 중 **GPSET0**과 **GPCLR0**을 통해 출력할 수 있다.
  그리고 GPIO 19번으로 입력되는 값은 GPLEV0~1 중 **GPLEV0**을 통해 읽을 수 있다.

  전체 설명)
  1. GPIO 18번 핀을 통해 LED를 끄고 키는 동작
     1.1. GPFSEL1 레지스터 값을 변경하여 GPIO 18번 핀을 Output 모드로 설정
     1.2. GPSET0 레지스터 값을 변경하여 GPIO 18번 핀에 High를 출력하여 LED 점등
     1.3. GPCLR0 레지스터 값을 변경하여 GPIO 18번 핀에 Low를 출력하여 LED 소등

  2. GPIO 19번 핀을 통해 스위치로 입력되는 값을 읽는 동작
     2.1. GPFSEL1 레지스터 값을 변경하여 GPIO 19번 핀을 Input 모드로 설정
     2.2. GPLEV0 레지스터 값을 통해 GPIO 19번 핀으로 입력되는 값(High 또는 Low)을 읽기

- ***GPFSEL1 주소 : 0x04**
  ***GPSET0 주소 : 0x1C**
  ***GPCLR0 주소 : 0x28**
  ***GPLEV0 주소 : 0x34**

  위 4개의 주소를 통해 각 레지스터로 접근할 수 있다.
  앞서 살펴보았던 Base 주소에 위 주소를 더해주면 4개의 레지스터 중 특정 레지스터로 접근할 수 있다.

  예를 들어, GPFSEL1로 접근하기 위해서는 Base 주소인 0x7e200000에 0x04를 더해주면 된다.
  즉 GPFSEL1에 접근하기 위해 사용되는 주소는 0x7e200004가 된다.

- GPFSEL1 레지스터 (주소 : 0x04)

Bits	Name	Description
0~2	GPIO 10	GPIO 10번 모드 설정
3~5	GPIO 11	GPIO 11번 모드 설정
6~8	GPIO 12	GPIO 12번 모드 설정
9~11	GPIO 13	GPIO 13번 모드 설정
12~14	GPIO 14	GPIO 14번 모드 설정
15~17	GPIO 15	GPIO 15번 모드 설정
18~20	GPIO 16	GPIO 16번 모드 설정
21~23	GPIO 17	GPIO 17번 모드 설정

24~26	**GPIO 18**	GPIO 18번 모드 설정
27~29	**GPIO 19**	GPIO 19번 모드 설정 Ex) 000 = Input mode 001 = Output mode
30~31	Reserved (사용되지 않는 공간)	

>> GPFSEL1은 32bit의 크기로 되어있으며, 한 GPIO당 모드 설정을 위해 3개의 bit 공간을 할당 받는다.
>> Raspberry Pi 4B는 32bit의 명령어형식을 갖는다. 그렇기 때문에 레지스터 하나의 크기는 32bit가 된다.
>> GPFSEL1 레지스터의 32개의 bit 중 24~26번째와 27~29번째 bit가 GPIO 18, 19번 핀의 모드 설정을 위해 사용되는
>> 공간이며 해당 bit 공간을 000으로 설정하게 되면 Input 모드, 001로 설정하게되면 Output 모드가 된다.

- GPSET0 레지스터 (주소 : 0x1C)

Bits	Name	Description
0~31	GPIO 0~31	0~31번에 해당하는 GPIO 핀 중 특정 핀 High 출력 Ex) 17번째 bit를 1로 Set시켜 주면 GPIO 18번 핀으로 High가 출력 됨 (bit는 0부터 시작함으로 17번째 bit는 18번 GPIO를 의미함)

>> 특정 GPIO 핀으로 High를 출력하기 위해서는 해당 핀이 Output 모드가 되어있어야 한다.

- GPCLR0 레지스터 (주소 : 0x28)

Bits	Name	Description
0~31	GPIO 0~31	0~31번에 해당하는 GPIO 핀 중 특정 핀 LOW 출력 Ex) 19번째 bit를 1로 Set시켜 주면 GPIO 18번 핀으로 Low가 출력 됨 (bit는 0부터 시작함으로 19번째 bit는 18번 GPIO를 의미함)

>> 특정 GPIO 핀으로 Low를 출력하기 위해서는 해당 핀이 Output 모드가 되어있어야 한다.

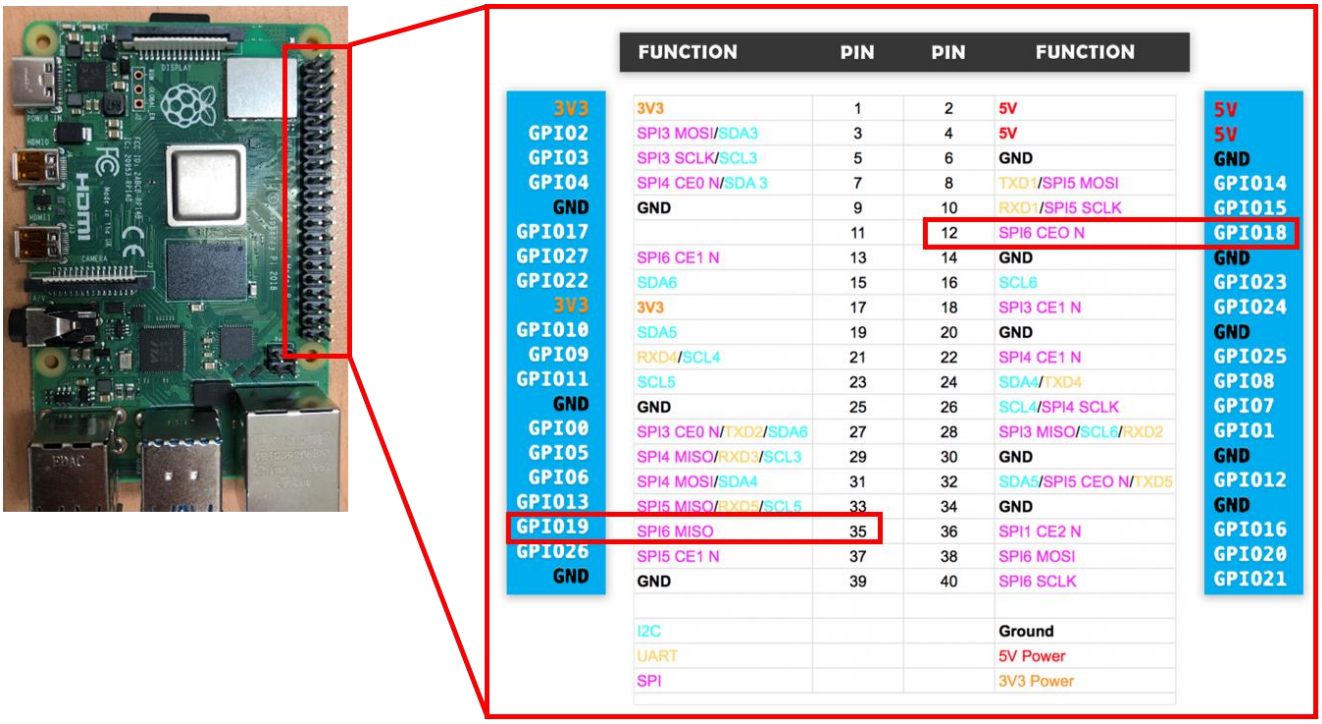
- GPLEV0 레지스터 (주소 : 0x34)

Bits	Name	Description
0~31	GPIO 0~31	0~31번에 해당하는 GPIO 핀 중 특정 핀 읽기 Ex) 19번째 bit가 1로 Set되어 있다면 GPIO 19번 핀으로 High가 입력되고 있는 것 (bit는 0부터 시작함으로 19번째 bit는 18번 GPIO를 의미함)

>> 특정 GPIO 핀으로 입력되는 값을 읽기 위해서는 해당 핀이 Input 모드가 되어있어야 한다.

Raspberry Pi에 LED와 슬라이드 스위치 연결

- Raspberry Pi 4B 핀맵

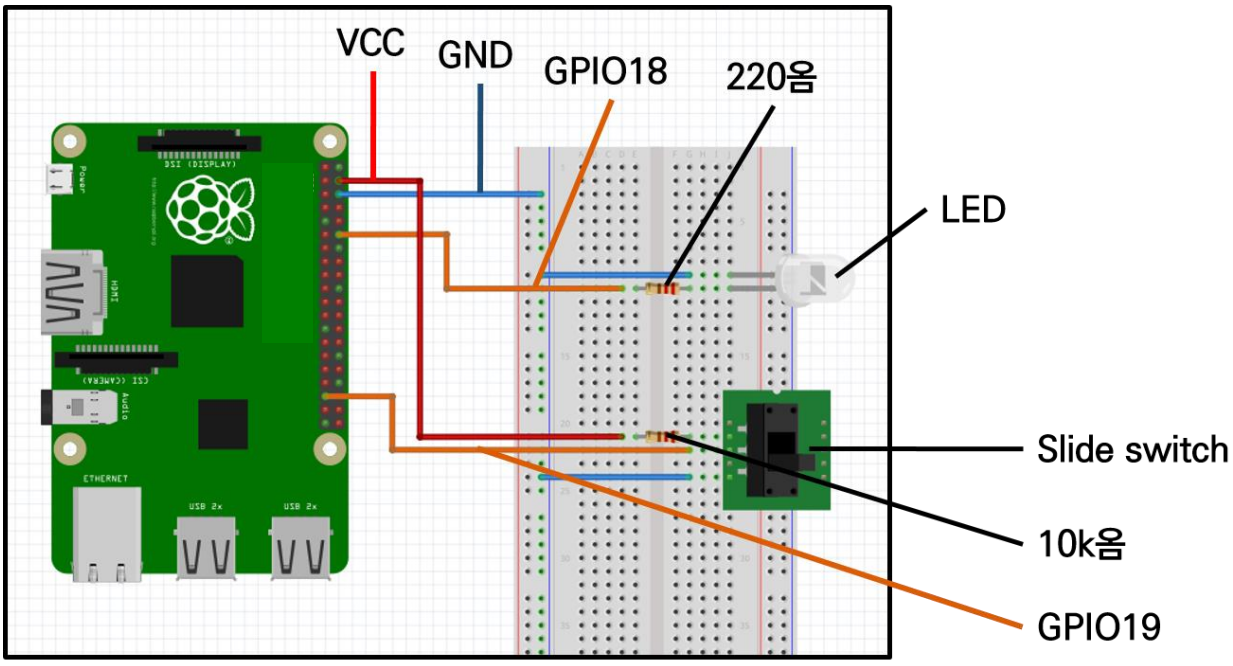


	FUNCTION	PIN	PIN	FUNCTION	
3V3	3V3	1	2	5V	5V
GPIO2	SPI3 MOSI/SDA3	3	4	5V	5V
GPIO3	SPI3 SCLK/SCL3	5	6	GND	GND
GPIO4	SPI4 CE0 N/SDA 3	7	8	TXD1/SPI5 MOSI	GPIO14
GND	GND	9	10	RXD1/SPI5 SCLK	GPIO15
GPIO17		11	12	SPI6 CEO N	GPIO18
GPIO27	SPI6 CE1 N	13	14	GND	GND
GPIO22	SDA6	15	16	SCL6	GPIO23
3V3	3V3	17	18	SPI3 CE1 N	GPIO24
GPIO10	SDA5	19	20	GND	GND
GPIO9	RXD4/SCL4	21	22	SPI4 CE1 N	GPIO25
GPIO11	SCL5	23	24	SDA4/TXD4	GPIO8
GND	GND	25	26	SCL4/SPI4 SCLK	GPIO7
GPIO0	SPI3 CE0 N/TXD2/SDA6	27	28	SPI3 MISO/SCL6/RXD2	GPIO1
GPIO5	SPI4 MISO/RXD3/SCL3	29	30	GND	GND
GPIO6	SPI4 MOSI/SDA4	31	32	SDA5/SPI5 CEO N/TXD5	GPIO12
GPIO13	SPI5 MISO/RXD5/SCL5	33	34	GND	GND
GPIO19	SPI6 MISO	35	36	SPI1 CE2 N	GPIO16
GPIO26	SPI5 CE1 N	37	38	SPI6 MOSI	GPIO20
GND	GND	39	40	SPI6 SCLK	GPIO21
	I2C			Ground	
	UART			5V Power	
	SPI			3V3 Power	

>> 본 실습에서는 Raspberry Pi 4B의 12번 핀과 35번 핀을 사용할 것이다.

>> 12번 핀과 35번 핀은 각각 GPIO 18, GPIO 19번 핀을 가리킨다.

- 회로 구성도



GPIO 디바이스 드라이버 코드 및 유저 프로그램 코드 살펴보기

1. gpio_dev.c (GPIO 디바이스 드라이버)

1.1. 헤더파일

```
#include <linux/module.h>      // 모듈 프로그래밍을 위한 헤더파일
#include <linux/kernel.h>      // 모듈 프로그래밍을 위한 헤더파일
#include <linux/init.h>        // 모듈 프로그래밍을 위한 헤더파일
#include <linux/io.h>          // RAM의 물리주소를 가상주소로 매핑시켜주는 함수인 ioremap()을 참조하는 헤더파일
#include <linux/fs.h>          // 캐릭터 디바이스 드라이버를 제작할 시 필요로한 헤더파일
#include <linux/uaccess.h>     // 커널공간과 유저공간의 데이터 송수신을 위한 copy_to_user() 함수를 참조하는 헤더파일
```

1.2. 매크로 및 전역변수

```
/*
 * GPIO Base 주소
 *
 * 라즈베리파이B+ 이하 : 0x20200000
 * 라즈베리파이2 이상 : 0x3F200000
 * 라즈베리파이4 이상 : 0xfE200000
 */
#define GPIO_BASE 0xFE200000

#define GPFSEL1 0x04    // GPFSEL1 주소
#define GPSET0 0x1C     // GPSET0 주소
#define GPCLR0 0x28     // GPCLR0 주소
#define GPLEV0 0x34     // GPLEV0 주소

static void __iomem *gpio_base;         // GPIO Base 주소를 가르키는 포인터 변수
volatile unsigned *gpfsel1;             // GPFSEL1 주소를 가르키는 포인터 변수
volatile unsigned *gpset0;              // GPSET0 주소를 가르키는 포인터 변수
volatile unsigned *gpclr0;              // GPCLR0 주소를 가르키는 포인터 변수
volatile unsigned *gplev0;              // GPLEV0 주소를 가르키는 포인터 변수
```

1.3. 파일 오퍼레이션

```
/* 디바이스 파일에 시스템 콜이 될 시 각 시스템 콜에 해당하는  함수들을 매핑 */
static struct file_operations dev_fops = {
        .read = dev_read,       // 디바이스 파일에 read 동작 발생 시 실행할 함수(dev_read)
        .open = dev_open,       // 디바이스 파일이 할당될 때 실행할 함수(dev_open)
        .release = dev_close    // 디바이스 파일이 해제될 때 실행할 함수(dev_close)
};
```

1.4. dev_init() 함수 : 레지스터 주소 매핑, GPIO 18&19 모드 설정, 캐릭터 디바이스 드라이버 생성

```
int __init dev_init(void){
        int result;

        printk("%s", __FUNCTION__);

        /* # ioremap()
         * 첫번째 인자로 GPIO의 물리주소를 입력하면
         * 가상주소로 매핑된 주소를 반환시켜준다.
         * 두번째 인자는 Page_size로 0x60으로 설정한다. */
        gpio_base = ioremap(GPIO_BASE, 0x60);

        /* gpio_base에 각 레지스터를 제어하는 주소들을 더하게되면
         * GPFSEL, GPSET등의 필드에 접근가능하다. */
        gpfsel1 = (volatile unsigned *)(gpio_base + GPFSEL1);
        gpset0 = (volatile unsigned *)(gpio_base + GPSET0);
        gpclr0 = (volatile unsigned *)(gpio_base + GPCLR0);
        gplev0 = (volatile unsigned *)(gpio_base + GPLEV0);

        /* GPIO 19는 INPUT, GPIO 18은 OUTPUT으로 설정 */
        *gpfsel1 = 0b 00 000 001 000 000 000 000 000 000 000 000;

        /* register_chrdev() 함수를 통해 새로운 캐릭터 디바이스 드라이버 등록 */
        result = register_chrdev(300, "gpio_dev", &dev_fops);
        if(result < 0)
                printk("[gpio_dev]register_chrdev failed..");
        else
                printk("[gpio_dev]register_chrdev successful..");

        return 0;
}
```

>> 해당 함수는 디바이스 드라이버 insmod 명령어를 통해 커널에 등록될 때 실행된다.

1.5. dev_open() : GPIO 18번 LED ON

```
int dev_open(struct inode *inode, struct file *filp){

        /* GPSET0필드의 19번째에 해당하는 18번 GPIO를 1로 set시켜 LED ON */
        *gpset0 = 0b00000000000001000000000000000000;

        /* GPCLR0필드는 모든 GPIO들을 set하지 않은 상태로 초기화 */
        *gpclr0 = 0b00000000000000000000000000000000;

        printk(KERN_INFO "[gpio_dev]%s : LED_ON", __FUNCTION__);
        return 0;
}
```

>> 해당 함수는 유저 프로그램에서 디바이스 파일을 열 때 실행된다.

1.6. dev_close() : GPIO 18번 LED OFF

```
int dev_close(struct inode *inode, struct file *filp){

        /* GPSET0필드의 모든 GPIO들을 set하지 않은 상태로 초기화 */
        *gpset0 = 0b00000000000000000000000000000000;

        /* GPCLR0필드의 19번째에 해당하는 18번 GPIO를 1로 set시켜 LED OFF */
        *gpclr0 = 0b00000000000001000000000000000000;

        printk(KERN_INFO "[gpio_dev]%s : LED_OFF", __FUNCTION__);

        return 0;
}
```

>> 해당 함수는 유저 프로그램에서 디바이스 파일을 닫을 때 실행된다.

1.7. dev_read() : GPIO 19번 READ

```
ssize_t dev_read(struct file *filp, char *buf, size_t count, loff_t *f_pos){
        int result;
        static const int HIGH = 1; /* 5v에 해당하는 HIGH(1)  */
        static const int LOW = 0; /* GND에 해당하는 LOW(0) */

        /* 20번째 bit를 1로 set시켜 GPLEV0의 20번째 bit와 &연산하여 결과값을 input에 저장 */
        unsigned int input = *gplev0 & 0b00000000000010000000000000000000;

        /* input이 0을 초과하는 값이면 5v에 해당하는 HIGH를 유저(App)으로 전달
         * input이 0이하일 시에는 GND에 해당하는 LOW를 유저(App)으로 전달 */
        if(input > 0){
                result = copy_to_user(buf, &HIGH, sizeof(HIGH));
        } else {
                result = copy_to_user(buf, &LOW, sizeof(LOW));
        }

        /* 유저영역으로 값이 잘 전달됐는지 확인하는 if */
        if(result == 0){
                printk(KERN_INFO "[gpio_dev]%s : SWITCH_READ", __FUNCTION__);
                printk(KERN_INFO "[gpio_dev]input : %d", input);
        } else {
                printk(KERN_INFO "[gpio_dev]%s : SWITCH_READ Failed(%d) ", __FUNCTION__, result);
        }

        return count;
}
```

>> 해당 함수는 유저 프로그램에서 디바이스 파일의 읽기 시스템 콜을 호출하였을 시 실행된다.

1.8. dev_exit() : 디바이스 드라이버를 커널에서 제거

```
/* 단지 캐릭터 디바이스 드라이버를 커널에서 제거하는 동작을 수행 */
void __exit dev_exit(void){
        printk("[gpio_dev]%s", __FUNCTION__);
        unregister_chrdev(300, "gpio_dev");
}
```

2. gpio_app.c (유저 프로그램)

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

int main(void) {
        int dev;
        int value;

        /* gpio_dev파일을 Read Write옵션으로 open한다.
         * GPIO드라이버의 open()함수가 실행되어 GPIO 18번의 LED가 ON된다. */
        dev = open("/dev/gpio_dev", O_RDWR);
        printf("%d\n", dev);

        while(1){
                /* gpio_dev파일을 read한다.
                 * GPIO드라이버의 read()함수가 실행되어 GPIO 19번의
                 * 데이터 값을 읽어서 value에 저장한다. */
                read(dev, &value, sizeof(value));
                printf("%d\n", value);

                sleep(1);
        }

        /* gpio_dev파일을 close한다.
         * GPIO드라이버의 close()함수가 실행되어 GPIO 18번의 LED가 OFF된다. */
        close(dev);
        return 0;
}
```

GPIO 디바이스 드라이버 모듈 소스코드 내려 받기 및 크로스 컴파일

- *본 실습에서는 타겟 시스템 커널의 버전을 5.10.x로 사용하였다.
  하지만 이번 실습을 따라하기 위해서 본 실습의 타겟 시스템의 커널 버전을 반드시 동일하게 맞출 필요는 없다.
  단지 호스트 시스템에 타겟 시스템의 커널 소스만 존재하면 된다.

1. GPIO 디바이스 드라이버 모듈 예제 코드 다운로드

1.1. 예제 코드를 받기 위해 Git 설치

명령어 : sudo apt-get install git

1.2. 저장소로부터 예제 코드 받기

명령어 : git clone https://github.com/INS-LAB/gpio-dev-driver.git

```
inslab-test-server@computer:~/git/5.10_rpi_kernel$ git clone https://github.com/INS-LAB/gpio-dev-driver.git
Cloning into 'gpio-dev-driver'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 2), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), 2.12 KiB | 362.00 KiB/s, done.
inslab-test-server@computer:~/git/5.10_rpi_kernel$ ls
char-dev-driver  gpio-dev-driver  kernel  linux  modules
```

>> git clone 명령어는 특정 저장소의 주소를 복사해오는 명령어이다.

>> 위 명령어를 실행하였다면 *gpio-dev-driver*라는 디렉터리가 하나 생성되었을 것이다.

2. 예제 코드 크로스 컴파일 진행

2.1. 다운로드 받은 예제 코드 디렉터리(gpio-dev-driver)로 이동

명령어 : cd gpio-dev-driver

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ ls
Makefile  gpio_app.c  gpio_dev.c
```

>> 해당 디렉터리 내에는 3개의 파일이 존재

>> *Makefile* : GPIO 디바이스 드라이버 파일(gpio_dev.c)을 컴파일 하기 위한 파일

>> *gpio_app.c* : GPIO 디바이스 드라이버를 사용하는 유저 프로그램 소스 코드 파일

>> *gpio_dev.c* : GPIO 디바이스 드라이버 예제 소스 코드 파일

2.2. 컴파일 전 Makefile 내용 수정

명령어 : vi Makefile

```
obj-m := gpio_dev.o

KDIR := # Please, insert the Rpi kernel directory path

CC := arm-linux-gnueabihf-gcc
OBJECT := gpio_app

default:
	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C$(KDIR) M=$(shell pwd) modules
	$(CC) -o $(OBJECT) $(OBJECT).c

clean:
	make -C$(KDIR) M=$(shell pwd) clean
	rm $(OBJECT)
```

>> 파일의 내용 중 KDIR 변수에 타겟 시스템의 커널 경로를 작성

>> 앞선 *"라즈베리파이 커널 컴파일"* 실습 때 사용한 디렉터리의 경로를 작성

```
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ ls
COPYING         MAINTAINERS     block    init                 modules.builtin          security  vmlinux.o
CREDITS         Makefile        certs    ipc                  modules.builtin.modinfo  sound     vmlinux.symvers
Documentation   Module.symvers  crypto   kernel               modules.order            tools
Kbuild          README          drivers  lib                  net                      usr
Kconfig         System.map      fs       mm                   samples                  virt
LICENSES        arch            include  modules-only.symvers scripts                  vmlinux
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ pwd
/home/insla-test-server/git/5.10_rpi_kernel/linux
```

>> 타겟 시스템 커널 소스코드의 디렉터리로 이동 후 pwd 명령어를 통해 경로 출력

>> 해당 커널 소스코드는 크로스 컴파일이 완료되어 있어야 한다.

>> 앞선 *"라즈베리파이 커널 컴파일"* 실습을 진행하였다면 당연히 컴파일이 완료된 상태일 것이다.

>> 이제 타겟 시스템 커널 소스코드 디렉터리의 경로를 알아냈으니 Makefile의 KDIR 변수에 입력하면 된다.

```
obj-m := gpio_dev.o

KDIR := /home/insla-test-server/git/5.10_rpi_kernel/linux

CC := arm-linux-gnueabihf-gcc
OBJECT := gpio_app

default:
	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C$(KDIR) M=$(shell pwd) modules
	$(CC) -o $(OBJECT) $(OBJECT).c

clean:
	make -C$(KDIR) M=$(shell pwd) clean
	rm $(OBJECT)
```

2.3. 컴파일 진행

명령어 : make

```
insla-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ make
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C/home/insla-test-server/git/5.10_rpi_kern
test-server/git/5.10_rpi_kernel/gpio-dev-driver modules
make[1]: Entering directory '/home/insla-test-server/git/5.10_rpi_kernel/linux'
  CC [M]  /home/insla-test-server/git/5.10_rpi_kernel/gpio-dev-driver/gpio_dev.o
  MODPOST /home/insla-test-server/git/5.10_rpi_kernel/gpio-dev-driver/Module.symvers
  CC [M]  /home/insla-test-server/git/5.10_rpi_kernel/gpio-dev-driver/gpio_dev.mod.o
  LD [M]  /home/insla-test-server/git/5.10_rpi_kernel/gpio-dev-driver/gpio_dev.ko
make[1]: Leaving directory '/home/insla-test-server/git/5.10_rpi_kernel/linux'
arm-linux-gnueabihf-gcc -o gpio_app gpio_app.c
insla-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ ls
Makefile        gpio_app    gpio_dev.c    gpio_dev.mod    gpio_dev.mod.o  modules.order
Module.symvers  gpio_app.c  gpio_dev.ko   gpio_dev.mod.c  gpio_dev.o
```

>> gpio-dev-driver 디렉터리에서 Makefile의 내용을 따라 컴파일 하기 위해 make 명령어 실행

>> 해당 명령어를 실행하게 되면 gpio_dev.ko와 gpio_app 파일이 생성된다.

>> gpio_dev.ko는 GPIO 디바이스 드라이버 커널 모듈 파일이며 gpio_app은 GPIO 디바이스 드라이버를 제어하는 유저 프로그램이다.

2.4. 모듈 정보 확인

명령어 : modinfo gpio_dev.ko

```
insla-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ modinfo gpio_dev.ko
filename:       /home/insla-test-server/git/5.10_rpi_kernel/gpio-dev-driver/gpio_dev.ko
license:        GPL
srcversion:     7EBD702059A56378046AFB4
depends:
name:           gpio_dev
vermagic:       5.10.95-v7l+ SMP mod_unload modversions ARMv7 p2v8
```

>> 모듈 정보를 확인해보면 ARM 코어의 5.10.95 커널에서 동작할 수 있는 모듈 임을 확인할 수 있다.

```
insla-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ uname -a
Linux computer 4.19.126-jihun #57 SMP Sat Feb 5 19:06:35 KST 2022 x86_64 x86_64 x86_64 GNU/Linux
```

>> 현재 호스트 시스템의 환경은 x86 코어의 4.19.126 커널이다.

>> 즉, gpio_dev.ko 모듈은 호스트 시스템과 맞지 않음으로 해당 모듈은 호스트 시스템에 등록할 수 없다.

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ sudo insmod gpio_dev.ko
insmod: ERROR: could not insert module gpio_dev.ko: Invalid module format
```

>> gpio_dev.ko를 호스트 시스템에 등록하기 위해 insmod 명령어를 사용하게 되면 에러를 반환하고 시스템에 등록하지 못하는 것을 확인할 수 있다.

>> gpio_app도 마찬가지로 타겟 시스템에 맞는 크로스 컴파일러를 통해 컴파일 되어 호스트 시스템에서는 사용할 수 없다.

3. GPIO 디바이스 드라이버 모듈을 타겟 시스템에 등록

3.1. GPIO 디바이스 드라이버 모듈과 유저 프로그램 이동

명령어 : scp gpio_dev.ko gpio_app pi@[RPI IP 주소]:/home/pi

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ scp gpio_dev.ko gpio_app pi@192.168.1.14:/home/pi
pi@192.168.1.14's password:
gpio_dev.ko                                   100% 6468    79.7KB/s   00:00
gpio_app                                      100% 8364    67.9KB/s   00:00
```

>> scp 명령어를 통해 타겟 시스템으로 모듈을 이동

>> scp 명령어 참조는 앞선 실습 내용인 *"툴체인"*을 참조하면 된다.

3.2. 타겟 시스템으로 원격 접속

명령어 : ssh pi@[RPI IP 주소]

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/gpio-dev-driver$ ssh pi@192.168.1.14
pi@192.168.1.14's password:
Linux raspberrypi 5.10.95-v7l+ #1 SMP Mon Feb 14 17:17:42 KST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar  2 04:13:47 2022
pi@raspberrypi:~ $
```

>> ssh 명령어를 통해 원격 접속이 성공하였다면 명령 프롬프트가 변경되었을 것이다.

```
pi@raspberrypi:~ $ ls
gpio_app  gpio_dev.ko
```

>> ls 명령어를 실행해보면 gpio_dev.ko와 gpio_app이 업로드 되어있는 것을 확인할 수 있다.

```
pi@raspberrypi:~ $ modinfo gpio_dev.ko
filename:       /home/pi/gpio_dev.ko
license:        GPL
srcversion:     7EBD702059A56378046AFB4
depends:
name:           gpio_dev
vermagic:       5.10.95-v7l+ SMP mod_unload modversions ARMv7 p2v8
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.10.95-v7l+ #1 SMP Mon Feb 14 17:17:42 KST 2022 armv7l GNU/Linux
```

>> modinfo 명령어와 uname –a 명령어를 통해 확인해보면 모듈의 커널 버전과 코어가 현재 타겟 시스템의 커널 버전과 코어와 동일할 것이다.

타겟 시스템에서 자신의 GPIO 디바이스 드라이버 사용해보기

1. 디바이스 드라이버 등록 및 디바이스 파일 생성

1.1. GPIO 디바이스 드라이버 모듈 등록

명령어 : sudo insmod gpio_dev.ko

```
pi@raspberrypi:~ $ ls
gpio_app  gpio_dev.ko
pi@raspberrypi:~ $ sudo insmod gpio_dev.ko
```

>> 해당 명령어를 실행한 뒤 아무런 내용이 뜨지 않는다면 정상 등록이 된 것이다.

```
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
gpio_dev               16384  0
cmac                   16384  3
algif_hash             16384  1
algif_skcipher         16384  1
af_alg                 28672  6 algif_hash
bnep                   20480  2
hci_uart               40960  1
btbcm                  16384  1 hci_uart
bluetooth             413696  26 hci_uart,
ecdh_generic           16384  2 bluetooth
```

>> lsmod를 통해 타겟 시스템에 등록된 모듈들을 출력해보면 gpio_dev가 등록되어 있는 것을 확인할 수 있다.

1.2. 디바이스 파일 생성

명령어 : sudo mknod /dev/gpio_dev c 300 0

```
pi@raspberrypi:~ $ sudo mknod /dev/gpio_dev c 300 0
pi@raspberrypi:~ $ ls -l /dev/gpio_dev
crw-r--r-- 1 root root 300, 0 Mar  2 11:06 /dev/gpio_dev
```

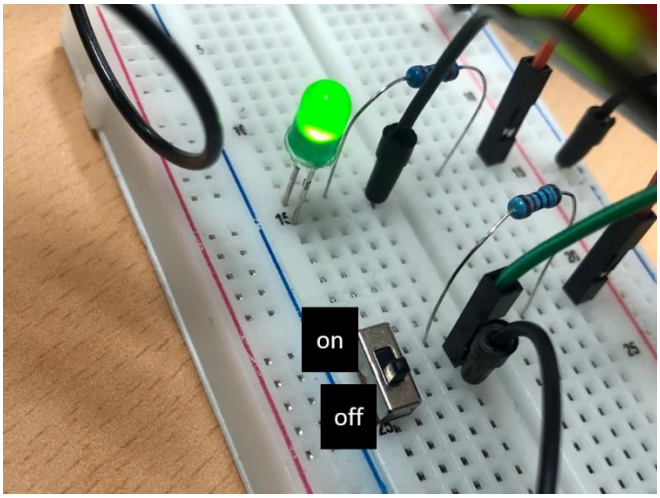
2. 유저 프로그램 실행

2.1. 유저 프로그램 실행

명령어 : sudo ./gpio_app

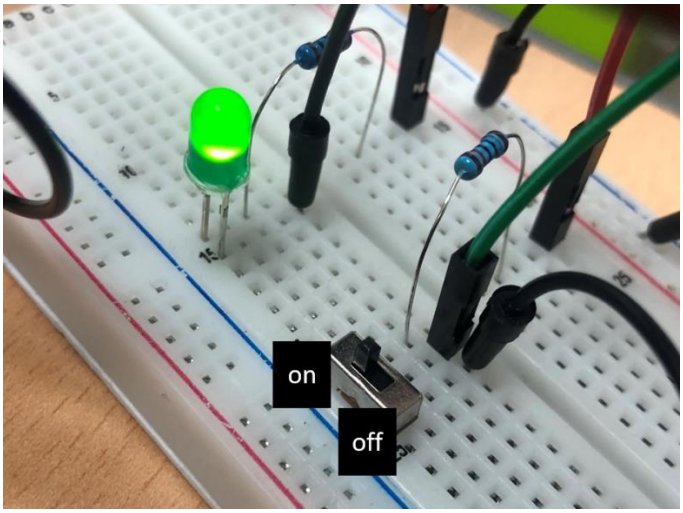
```
pi@raspberrypi:~ $ ls
gpio_app  gpio_dev.ko
pi@raspberrypi:~ $ sudo ./gpio_app
open successful!
0
0
0
0
0
1
1
1
1
1
0
0
```

>> 유저 프로그램이 실행된 모습



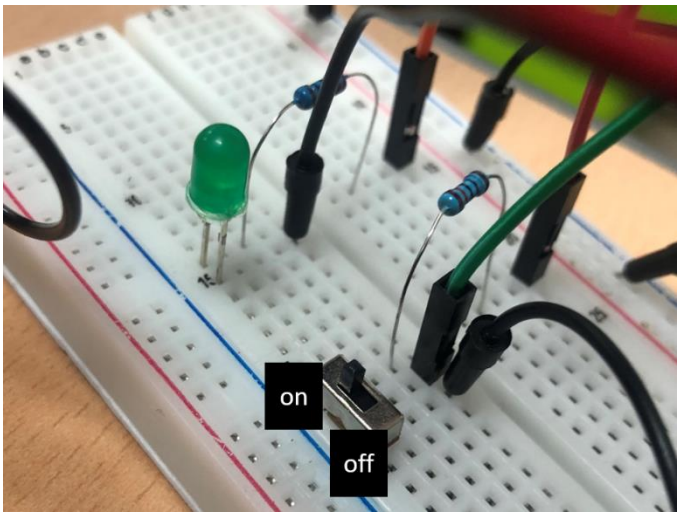
>> 유저 프로그램이 실행되어서 디바이스 파일을 open하여 LED가 켜지게 됨

>> 스위치가 OFF(LOW값이 들어가는 상태)에 위치해 있어 터미널에 0이 찍힘



>> 유저 프로그램이 실행되어서 디바이스 파일을 open하여 LED가 켜지게 됨

>> 스위치가 ON(HIGH값이 들어가는 상태)에 위치해 있어 터미널에 1이 찍힘



>> 유저 프로그램이 종료되면서 디바이스 파일이 close되어 LED가 꺼지게 됨

>> 스위치가 ON(HIGH값이 들어가는 상태)에 위치해 있어도 터미널에는 아무런 값이 찍히지 않음