



EDDI

Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정

제 1기

2021. 12. 17

박성환

CONTENTS

1. GPIO를 통한 LED 제어 및 인터럽트 수신
2. 2. I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

CONTENTS

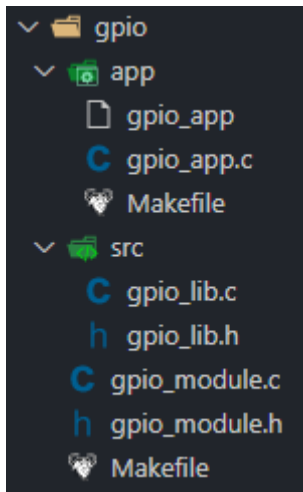
GPIO를 통한 LED 제어 및 인터럽트 수신

서론

- 리눅스에서 GPIO를 제어하는 방법에는 여러가지가 있음
 - ① sys/class에서 gpio export해서 사용하는 방법
 - ② gpio driver 구현하여 직접 컨트롤 하는 방법(커널직접/모듈)
 - ③ 디바이스 트리를 사용하는 방법
- 위의 서론이 맞는지 정확히는 모르겠으나 우선 찾아본 바로는 그렇고
1번은 간단히 터미널에서 실제 포트가 맞는지 확인용으로만 사용하고
2번을 사용해서 구현할 예정

파일 구성

- app : 실제 동작하는 main 함수로 구성
- src : 라이브러리 함수 구현
- module : gpio driver 코드 구현



모듈 관련 명령어

- insmod : 커널 모듈 load
- rmmod : 커널 모듈 unload
- lsmod : 현재 커널에 올라와 있는 모듈 리스트
- modinfo : 현재 모듈에 대한 정보를 출력
- ls -l /dev : 현재 시스템에서 사용하고 있는 device(c/b, 주/부번호 파악 가능)
- mknod : 디바이스 파일 생성
- dmesg : 커널 메시지로 디바이스 드라이버 디버깅할 때 유용

CONTENTS

GPIO를 통한 LED 제어 및 인터럽트 수신

GPIO_MODULE

- 아래 순서대로 모듈 코드 구현
- ① #include <linux/module.h> 등 추가
- ② MODULE 정보 입력
- ③ init/exit 구현
- ④ file_operations 함수 포인터 매칭
- ⑤ ioctl을 이용한 led 제어 코드 구현
- ⑥ 인터럽트 동작 구현
- ⑦ 드라이버 및 디바이스 파일 로딩

- ② MODULE 정보 입력
 - 모듈 라이선스, 기본 정보 기입
 - modinfo 로 아래 정보(해당 모듈의 버전이나 정보)를 알 수 있음

```
MODULE_LICENSE("GPL");  
MODULE_AUTHOR("Seonghwan Park");  
MODULE_DESCRIPTION("GPIO input/output control");  
MODULE_INFO(board, "Jetson AGX Xavier 4.9.253-tegra");
```

- ③ init/exit 구현
 - 커널 모듈 등록과 해제
 - 사용자 함수 gpiodrv_init, exit 구현 후 아래에 함수기입

```
module_init(gpiodrv_init);  
module_exit(gpiodrv_exit);
```

- ④ file operations 함수 포인터 매칭
 - 애플리케이션과의 저수준 입출력을 위해 file_operation 구조체 존재
 - 함수 포인터로 매칭하여 사용

```
/* Match File Operation functions */  
struct file_operations gpio_fops = {  
    .open = gpiodrv_open,  
    .release = gpiodrv_release,  
    .write = gpiodrv_write,  
    .read = gpiodrv_read,  
    .unlocked_ioctl = gpiodrv_ioctl,  
};
```

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

③ init/exit 구현(init)

```
static int __init gpdrv_init(void)
{
    int ret;

    printk("GPIO Module is up... \n");

    if((result = gpdrv_register_cdev() < 0)
    {
        return result;
    }

    #if 1 /* Interrupt initialize */
    gpio_request(GPIO_PIN_249, "DOOR");
    door_irq = gpio_to_irq(GPIO_PIN_249);
    if( request_irq(door_irq, (void*)door_isr_func,
        IRQF_TRIGGER_RISING, "DOOR", NULL))
    {
        printk("failed to request external interrupt.\n");
        ret = -ENOENT;
        return ret;
    }
    #endif

    return 0;
}
```

```
static int gpdrv_register_cdev(void)
{
    int error;

    /* allocation device number */
    if(gpdrv_major) {
        gpdrv_dev = MKDEV(gpdrv_major, gpdrv_minor);
        error = register_chrdev_region(gpdrv_dev, 1, "sk");
        주 번호 알때
    }
    else {
        error = alloc_chrdev_region(&gpdrv_dev, gpdrv_minor, 1, "sk");
        gpdrv_major = MAJOR(gpdrv_dev); 주 번호 동적할당
    }

    if(error < 0) {
        printk(KERN_WARNING "gpio : can't get major %d\n", gpdrv_major);
        return result;
    }
    printk("major number=%d\n", gpdrv_major);
    (초기화할 cdev 구조체, 등록될 fop 포인터)
    cdev_init(&gpdrv_cdev, &gpio_fops);
    gpdrv_cdev.owner = THIS_MODULE;
    gpdrv_cdev.ops = &gpio_fops;
    error = cdev_add(&gpdrv_cdev, gpdrv_dev, 1);
    준비된 cdev 구조체를 커널에 등록
    if(error)
        printk(KERN_NOTICE "gpio Register Error %d\n", error);

    return 0;
}
```

- 디바이스 드라이버를 사용하기 위해서는 커널에 등록하는 절차 필요
- 커널에 등록할 때는 주번호를 사용하여 등록이 필요
- 주번호는 이미 알고 있거나 동적할당이 가능

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

- ⑤ ioctl 함수를 이용한 led구현(자세히)
- 일반적으로 I/O Control에 관련한 작업을 수행하는 함수
- 대부분의 ioctl 메소드 구현은 cmd 인수값에 따라 올바른 동작을 수행하는 switch 문으로 구성함

init 함수 내부(초기화)

```
/* 0.initialize GPIO ports */
gpio_request(GPIO_PIN_424, "LED1");
gpio_request(GPIO_PIN_393, "LED2");
gpio_request(GPIO_PIN_256, "PHOTOSENS1");
gpio_request(GPIO_PIN_251, "PHOTOSENS2");
gpio_request(GPIO_PIN_250, "SOLENOID");
gpio_request(GPIO_PIN_289, "PCIE_12V");

/* 1.off the gpio pin value */
gpio_set_value(GPIO_PIN_424, 0);
gpio_set_value(GPIO_PIN_393, 0);
gpio_set_value(GPIO_PIN_250, 0);
gpio_set_value(GPIO_PIN_289, 0);
//gpio_set_value(GPIO_PIN_344, 0);

/* 2. set the control register to output */
gpio_direction_output(GPIO_PIN_424, 0);
gpio_direction_output(GPIO_PIN_393, 0);
gpio_direction_output(GPIO_PIN_250, 0);
gpio_direction_output(GPIO_PIN_289, 0);

/* 3. set the input register */
gpio_direction_input(GPIO_PIN_256);
gpio_direction_input(GPIO_PIN_251);
```

```
#define GPIO_MAGIC 'k'
#define GPIO_MAXNR 6

typedef struct {
    unsigned int pinNumber;
    bool state;
} __attribute__((packed)) ioctl_info;
```

```
#define GPIO_SET_LED1 _IO(GPIO_MAGIC, 0)
#define GPIO_GET_LED1 _IO(GPIO_MAGIC, 1)
#define GPIO_SET_LED2 _IO(GPIO_MAGIC, 2)
#define GPIO_GET_LED2 _IO(GPIO_MAGIC, 3)
#define GPIO_SET_SOLENOID _IO(GPIO_MAGIC, 4)
#define GPIO_GET_SOLENOID _IO(GPIO_MAGIC, 5)
#define GPIO_GET_DOOR _IO(GPIO_MAGIC, 6)
#define GPIO_GET_PHOTOSENSOR1 _IO(GPIO_MAGIC, 7)
#define GPIO_GET_PHOTOSENSOR2 _IO(GPIO_MAGIC, 8)
#define GPIO_SET_PCIE _IO(GPIO_MAGIC, 9)
#define GPIO_GET_PCIE _IO(GPIO_MAGIC, 10)
```

```
static long gpiodrv_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
```

애플리케이션에서 ioctl 함수 호출

```
{
    // sk_info ctrl_info;
    int size, err;
    int sts = 0;

    if 1
    if(_IOC_TYPE(cmd) != GPIO_MAGIC) {
        printk("_IOC_TYPE error\n");
        return -EINVAL;
    };
    if(_IOC_NR(cmd) >= GPIO_MAXNR) {
        printk("_IOC_NR error\n");
        return -EINVAL;
    };

    size = _IOC_SIZE(cmd);

    // POINT: check commands
    if (size) {
        err = 0;
        if(_IOC_DIR(cmd) & _IOC_READ)
            err = verify_area(VERIFY_WRITE, (void *)arg, size);
        err = access_ok(VERIFY_WRITE, (void *)arg, size);
        else if (_IOC_DIR(cmd) & _IOC_WRITE)
            err = verify_area(VERIFY_READ, (void *)arg, size);
            err = access_ok(VERIFY_READ, (void *)arg, size);
        if (err) return err;
    }
}
#endif
```

cmd 유효성 검사

```
// allows to access device using ioctl commands
switch(cmd) {
    case GPIO_SET_LED1:
        printk("GPIO_SET_LED1\n");
        gpio_set_value(GPIO_PIN_424, arg);
        break;
    case GPIO_GET_LED1:
        printk("GPIO_GET_LED1\n");
        sts = gpio_get_value(GPIO_PIN_424);
        break;
    case GPIO_SET_LED2:
        printk("GPIO_SET_LED2\n");
        gpio_set_value(GPIO_PIN_393, arg);
        break;
    case GPIO_GET_LED2:
        printk("GPIO_GET_LED2\n");
        sts = gpio_get_value(GPIO_PIN_393);
        break;
    case GPIO_GET_DOOR:
        printk("GPIO_GET_DOOR\n");
        sts = gpio_get_value(GPIO_PIN_249);
        break;
    case GPIO_SET_PCIE:
        printk("GPIO_SET_PCIE\n");
        gpio_set_value(GPIO_PIN_289, arg);
        break;
    case GPIO_GET_PCIE:
        printk("GPIO_GET_PCIE\n");
        sts = gpio_get_value(GPIO_PIN_289);
        break;
    default:
        printk("Can't find any cmd\n");
        sts = -1;
        break;
}
```

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

⑥ 인터럽트 동작 구현(App 영역에 시그널 전달 포함)

gpio_app.c(사용자)

```
⑤ #if 1
void signal_handler(int sigum)
{
    if(gpio_getValue(Door_sensor1))
        printf("Door_sensor1 is Closed\n");
    #if 0
        if(sigum == SIGIO) { /* SIGIO 신호면 종료 빠져나가기 */
            printf("SIGIO\n");
            exit(1);
        }
    #endif
    시그널 처리를 위한 핸들러 등록
    SIGIO 신호가 오면 signal_handler 처리
}

② signal(SIGIO, signal_handler); //for Interrupt

#if 1 /* only send pid */
    sprintf(buf, "%d", getpid());
    write(fd, buf, strlen(buf)); //send the pid of this process
#else /* send siginfo to the process */
    // sprintf(buf, "%d\n", sigum);
    // write(fd, buf, strlen(buf));
#endif

#if 1 /* read pid */
    if(read(fd, buf, strlen(buf)) == 0)
        printf("Got PID Number %s\n", buf);
#endif
```

gpio_module.c(드라이버)

```
③ /* when signal occurred, register PID to send */
pid = simple_strtol(pidstr, &endptr, 10); //string to long
if(endptr != NULL) { 시그널 발생시 보낼 프로세서 ID 등록
    task = pid_task(find_vpid(pid), PIDTYPE_PID);
    if(task == NULL) {
        printk("Error: Can't find PID from user application\n");
        return 0;
    }
}

① #if 1 /* Interrupt initialize */
gpio_request(GPIO_PIN_249, GPIO 핀번호 IRQ번호로 매핑
door_irq = gpio_to_irq(GPIO_PIN_249);
if(request_irq(door_irq, (void*)door_isr_func,
    IRQF_TRIGGER_RISING, "DOOR", NULL)) {
    커널에 IRQ요청하여 핸들러 등록
    printk("failed to request external interrupt.\n");
    ret = -ENOENT;
    return ret;
}
#endif

/* Implementation of functions */
④ static irqreturn_t door_isr_func(int irq, void *dev_id)
{
    인터럽트 서비스 루틴
    if(irq == door_irq && gpio_get_value(GPIO_PIN_249)) {
    #if 1
        static struct siginfo sinfo; /* struct for siginfo */
        memset(&sinfo, 0, sizeof(struct siginfo));
        sinfo.si_signo = SIGIO;
        sinfo.si_code = SI_USER;
        send_sig_info(SIGIO, &sinfo, task); /* send the signal to the process */
    #else
        sys_kill(pid, SIGINT); //sys kill is not working well
    #endif
    }
```

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

⑦ 드라이버 및 디바이스 파일 로딩

1) gpio module 빌드

- make 명령어로 Makefile 규칙대로 컴파일 실행
- xxx.ko 드라이버 파일을 포함한 여러 파일들이 생성됨

```
/jetson/custom_drivers/gpio$ ls
app gpio_module.c gpio_module.h Makefile src
/jetson/custom_drivers/gpio$ make
make -C /lib/modules/4.9.253-tegra/build/ M=/home/gone/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio modules
make[1]: Entering directory '/usr/src/linux-headers-4.9.253-tegra-ubuntu18.04_aarch64/kernel-4.9'
CC [M] /home/gone/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio/gpio_module.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/gone/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio/gpio_module.mod.o
LD [M] /home/gone/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio/gpio_module.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.9.253-tegra-ubuntu18.04_aarch64/kernel-4.9'
/jetson/custom_drivers/gpio$ ls
app gpio_module.c gpio_module.h gpio_module.ko gpio_module.mod.c gpio_module.mod.o gpio_module.o Makefile modules.order Module.symvers src
```

모듈 빌드 시 생성되는 파일
gpio_module.ko 파일이 드라이버 파일임

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

⑦ 드라이버 및 디바이스 파일 로딩

2) gpio module load

- insmod로 gpio module load(MODULE_INIT() 동작)
- Major number(주번호) 505번이 동적할당됨을 확인함
- xxx.ko 드라이버 파일을 포함한 여러 파일들이 생성됨

```
/jetson/custom_drivers/gpio$ sudo insmod gpio_module.ko
/jetson/custom_drivers/gpio$ dmesg | tail -10
[252213.214138] tegra-i2c 31e0000.i2c: no acknowledge from address 0x76
[252273.281346] tegra-i2c 31e0000.i2c: no acknowledge from address 0x77
[278229.259948] The GPIO module is down...
[278264.685444] GPIO Module is up...
[278264.685560] major number=505
[278264.685827] gpio tegra-gpio-aon wake42 for gpio=1(AA:1)
[278279.451016] The GPIO module is down...
[278288.747228] GPIO Module is up...
[278288.747342] major number=505
[278288.747582] gpio tegra-gpio-aon wake42 for gpio=1(AA:1)
```

MODULE_INIT() 매크로 동작하여 초기화 및 디바이스 커널 등록됨

CONTENTS

GPIO를 통한 input/output 및 인터럽트 제어하기

⑦ 드라이버 및 디바이스 파일 로딩

3) device 파일 생성

- 사용자는 device file을 이용해 장치를 사용할 수 있음
- device driver로 사용할 장치 파일을 만드는 것
- 장치 파일은 사용자와 device driver를 연결해 주는 매개체

```
gone@gone-jetson:~/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio$ sudo mknod /dev/GPIODEV c 505 0
gone@gone-jetson:~/Projects/slideScanner/luceon_project/jetson/custom_drivers/gpio$ ls -al /dev/GPIO*
crw-r--r-- 1 root root 505, 0 12월 17 15:52 /dev/GPIODEV
```

CONTENTS

1. GPIO를 통한 input/output 및 인터럽트 제어하기
2. I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

서론

- jetson에서 제공하는 커널에 기본 i2c 드라이버를 제공해줌
- 사용자 영역에서 파일 컨트롤로 쉽게 제어 가능함

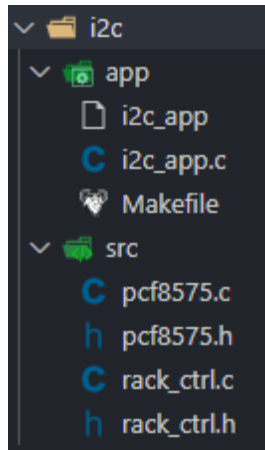
```
gone@gone-jetson:~$ ls -l /dev/i2c*  
crw-rw---- 1 root i2c 89, 0 12월 14 10:32 /dev/i2c-0  
crw-rw---- 1 root i2c 89, 1 12월 14 10:32 /dev/i2c-1  
crw-rw---- 1 root i2c 89, 2 12월 14 10:32 /dev/i2c-2  
crw-rw---- 1 root i2c 89, 3 12월 14 10:32 /dev/i2c-3  
crw-rw---- 1 root i2c 89, 4 12월 14 10:32 /dev/i2c-4  
crw-rw---- 1 root i2c 89, 5 12월 14 10:32 /dev/i2c-5  
crw-rw---- 1 root i2c 89, 6 12월 14 10:32 /dev/i2c-6  
crw-rw---- 1 root i2c 89, 7 12월 14 10:32 /dev/i2c-7  
crw-rw---- 1 root i2c 89, 8 12월 14 10:32 /dev/i2c-8
```

사전 준비

- 아래 패키지 설치 필요
sudo apt-get install i2c-tools
sudo apt-get install libi2c-dev

폴더 구성

- app : 실제 동작하는 main 함수로 구성
- src : 라이브러리 함수 구현



CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

i2c 핀맵

Jetson AGX Xavier Expansion Header					
Sysfs GPIO	Connector Label	Pin	Pin	Connector Label	Sysfs GPIO
	3.3 VDC Power, 1A max	1	2	5.0 VDC Power, 1A max	
	I2C_GP5_DAT General I2C #5 Data 1.8/3.3V, I2C Bus 8	3	4	5.0 VDC Power, 1A max	
	I2C_GP5_CLK General I2C #5 Clock 1.8/3.3V, I2C Bus 8	5	6	GND	
gpio422	MCLK05 Audio Master Clock 1.8/3.3V	7	8	UART1_TX UART #1 Transmit 3.3V	
	GND	9	10	UART1_RX UART #1 Receive 3.3V	

i2c 장치 확인

- 디바이스 파일 확인 : `ls -al /dev/i2c*`

```
/jetson$ ls -al /dev/i2c*
crw-rw---- 1 root i2c 89, 0 12월 14 10:32 /dev/i2c-0
crw-rw---- 1 root i2c 89, 1 12월 14 10:32 /dev/i2c-1
crw-rw---- 1 root i2c 89, 2 12월 14 10:32 /dev/i2c-2
crw-rw---- 1 root i2c 89, 3 12월 14 10:32 /dev/i2c-3
crw-rw---- 1 root i2c 89, 4 12월 14 10:32 /dev/i2c-4
crw-rw---- 1 root i2c 89, 5 12월 14 10:32 /dev/i2c-5
crw-rw---- 1 root i2c 89, 6 12월 14 10:32 /dev/i2c-6
crw-rw---- 1 root i2c 89, 7 12월 14 10:32 /dev/i2c-7
crw-rw---- 1 root i2c 89, 8 12월 14 10:32 /dev/i2c-8
```

- 장치 연결 확인 : `i2cdetect -r 8` (r: return 값, 8: 디바이스 파일 번호)

```
/jetson$ i2cdetect -r 8
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-8 using receive byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n]
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20: 20 21 22 -- 24 25 --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

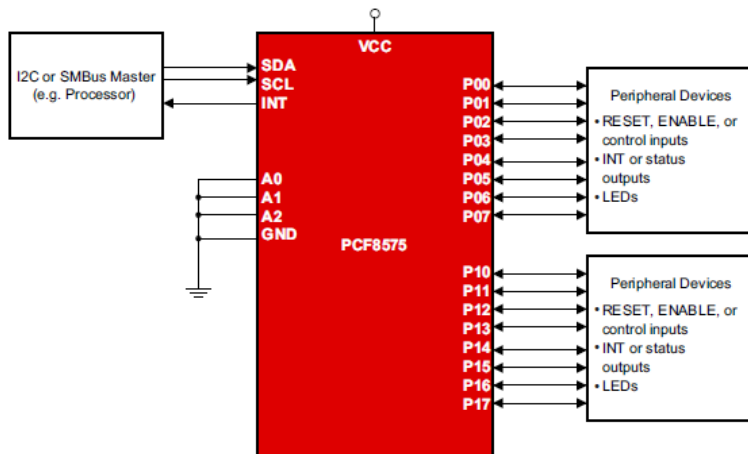
CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

PCF8575 datasheet 확인

- 총 16개의 Input/Output 포트 제공

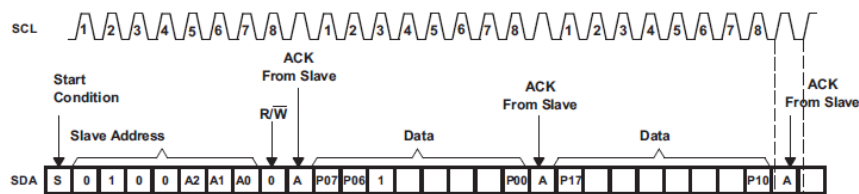
Simplified Schematic



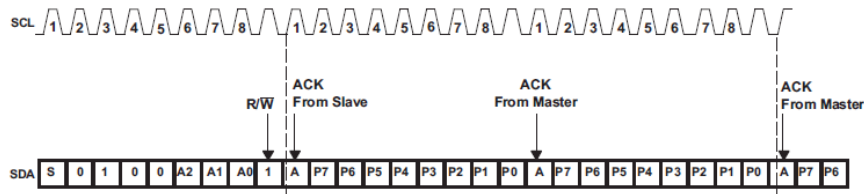
Address

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I ² C slave address	L	H	L	L	A2	A1	A0	R/W
P0x I/O data bus	P07	P06	P05	P04	P03	P02	P01	P00
P1x I/O data bus	P17	P16	P15	P14	P13	P12	P11	P10

Write



Read

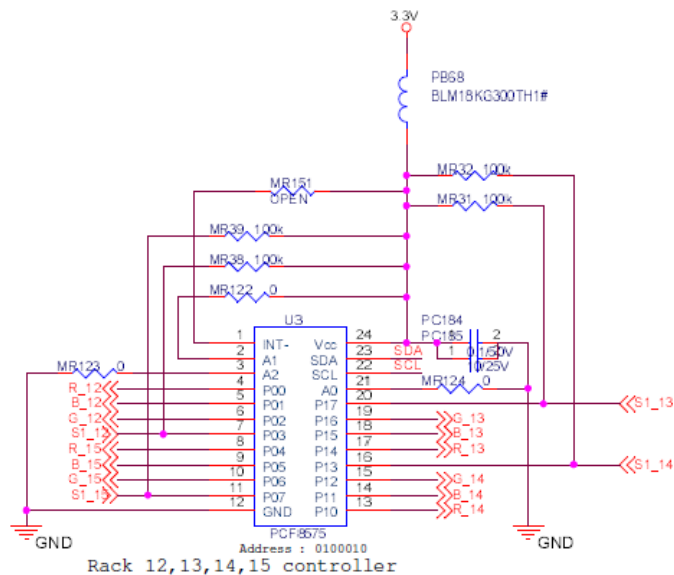


CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

회로 구성

- Output : R,G,B LED 사용
- Input : 스위치 사용



CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

PCF8575 코드 구현

Open

```
bool PCF8575_open(void)
{
    char fileNameBuffer[32];

    sprintf(fileNameBuffer, "/dev/i2c-%d", kI2CBus);
    kI2CFileDescriptor = open(fileNameBuffer, O_RDWR);
    if (kI2CFileDescriptor < 0) 아미 만들어져 있는 device file open
    {
        // Could not open the file
        error = errno;
        return -1;
    }
    if (ioctl(kI2CFileDescriptor, I2C_SLAVE, kI2CAddress) < 0) {
        // Could not open the device on the bus
        error = errno;
        return -1;
    }

    //printf("PCF8575 Device Address: 0x%02X\n", kI2CAddress);
    //printf("PCF8575 Device Bus: 0x%02X\n", kI2CBus);
    return 0;
}
```

Close

```
void PCF8575_close(void)
{
    if (kI2CFileDescriptor > 0) {
        close(kI2CFileDescriptor);
        // WARNING - this is not quite right, need to check for error first
        kI2CFileDescriptor = -1;
        printf("PCF8575 BUS closed : 0x%02X\n", kI2CBus);
    }
}
```


CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

PCF8575 코드 구현

Write

```
int PCF8575_write(int addr, int writeValue)
{
    unsigned char buf[2];
    int toReturn;

    ioctl(kI2CFileDescriptor, I2C_SLAVE_FORCE, addr);

    buf[1] = (writeValue & 0xFF00) >> 8;
    buf[0] = writeValue & 0xFF;

    if ((toReturn = write(kI2CFileDescriptor, buf, 2)) < 0) {
        fprintf(stderr, "writing error! (%X): %s\r\n", writeValue, strerror(errno));
        exit(EXIT_FAILURE);
    }
    // return -1;

    return toReturn;
}
```

Read

```
int PCF8575_read(int addr)
{
    int toReturn;
    int tmpData;
    unsigned char buf[2];
    int returnData;

    ioctl(kI2CFileDescriptor, I2C_SLAVE_FORCE, addr);

    if ((toReturn = read(kI2CFileDescriptor, buf, 2)) < 0) {
        fprintf(stderr, "reading error! (%X): %s\r\n", addr, strerror(errno));
        exit(EXIT_FAILURE);
    }
    // return -1;

    tmpData = buf[0] + (buf[1] << 8);
    // fprintf(stdout, "tmpData = 0x%04X\r\n", tmpData);

    /* Before Reading, should input Port set to high */
    tmpData = tmpData | ((0x1<<15) + (0x1<<11) + (0x1<<7) + (0x1<<3));
    // fprintf(stdout, "tmpData = 0x%04X\r\n", tmpData);

    PCF8575_write(addr, tmpData);

    if ((toReturn = read(kI2CFileDescriptor, buf, 2)) < 0) {
        fprintf(stderr, "Reading error! (%X): %s\r\n", addr, strerror(errno));
        exit(EXIT_FAILURE);
    }
    // return -1;

    returnData = buf[0] + (buf[1] << 8);
    // fprintf(stdout, "Read 2byte = 0x%04X\r\n", returnData);

    return returnData;
}
```

CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

Rack 코드 컨셉

- Rack 관련 정보를 구조체로 표현
- 배열로 정보를 구성하여 해당 정보를 찾는 컨셉

```
typedef enum _Rack_Color {  
    ... WHITE = 0x0,  
    ... RED = 0x6,  
    ... BLUE = 0x5,  
    ... GREEN = 0x3,  
    ... OFFLED = 0x7  
} Rack_Color;
```

```
typedef struct _Rack_State {  
    ... Rack_Color led;  
    ... bool sw;  
} Rack_State;  
  
typedef struct _Rack_Info {  
    ... const Rack_Num num;  
    ... const unsigned char addr;  
    ... const unsigned int posParam;  
    ... Rack_State state;  
} Rack_Info;
```

```
typedef enum _Rack_Num {  
    ... RACK_1,  
    ... RACK_2,  
    ... RACK_3,  
    ... RACK_4,  
    ... RACK_5,  
    ... RACK_6,  
    ... RACK_7,  
    ... RACK_8,  
    ... RACK_9,  
    ... RACK_10,  
    ... RACK_11,  
    ... RACK_12,  
    ... RACK_13.
```

```
Rack_Info RackInfo[] = {  
    ... {RACK_1, 0x21, 0, {OFFLED, DISCONN}}, ... {RACK_2, 0x20, 8, {OFFLED, DISCONN}}, ... {RACK_3, 0x20, 12, {OFFLED, DISCONN}},  
    ... {RACK_4, 0x20, 0, {OFFLED, DISCONN}}, ... {RACK_5, 0x20, 4, {OFFLED, DISCONN}}, ... {RACK_6, 0x24, 0, {OFFLED, DISCONN}},  
    ... {RACK_7, 0x24, 4, {OFFLED, DISCONN}}, ... {RACK_8, 0x24, 8, {OFFLED, DISCONN}}, ... {RACK_9, 0x24, 12, {OFFLED, DISCONN}},  
    ... {RACK_10, 0x21, 12, {OFFLED, DISCONN}}, ... {RACK_11, 0x21, 8, {OFFLED, DISCONN}}, ... {RACK_12, 0x22, 0, {OFFLED, DISCONN}},  
    ... {RACK_13, 0x22, 12, {OFFLED, DISCONN}}, ... {RACK_14, 0x22, 8, {OFFLED, DISCONN}}, ... {RACK_15, 0x22, 4, {OFFLED, DISCONN}},  
    ... {RACK_16, 0x25, 12, {OFFLED, DISCONN}}, ... {RACK_17, 0x25, 8, {OFFLED, DISCONN}}, ... {RACK_18, 0x25, 4, {OFFLED, DISCONN}}  
};
```

CONTENTS

I2C통신을 사용한 Expansion IO (PCF8575) 제어하기

Rack 코드 컨셉

- Rack 관련 정보를 구조체로 표현
- 배열로 정보를 구성하여 해당 정보를 찾는 컨셉

```
int Rack_setLed(Rack_Num rackNum, Rack_Color rackColor)
{
    int i;
    int data;
    int toReturn;

    for(i = 0; i < RACK_CNT; i++)
    {
        if(RackInfo[i].num == rackNum)
        {
            data = PCF8575_read(RackInfo[i].addr);
            RackInfo[i].state.led = rackColor;
            data = (data & ~(0x7 << RackInfo[i].posParam)) | (rackColor << RackInfo[i].posParam);
            toReturn = PCF8575_write(RackInfo[i].addr, data);
            if(toReturn < 0) {
                printf("Cannot set the led \n");
                return -1;
            }
            return toReturn;
        }
    }

    printf("Cannot find the rack %d\n", rackNum);
    return -1;
}
```