

Embedded System Software 과제 2

(과제 수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20181664, 이동건

개발기간: 2023. 05. 10. – 2023. 05. 17.

최 종 보 고 서

I. 개발 목표

- 각 과제마다 주어지는 주제를 바탕으로 본 과제에서 추구하는 개발 목표를 설정하고 그 내용을 기술할 것.
- Kernel timer 활용 방법과 fpga hw 조작 방법, module 생성 방법 을 활용하여 명세서에 명세되어 있는 요구사항을 만족하는 device driver 를 모듈 형태로 구현한다
- 해당 device driver 를 직접 실행하는 유저 프로그램 도 함께 만드는 것을 목표로 한다.

II. 개발 범위 및 내용

- 자신들이 설계한 개발 목표를 달성하기 위하여 어떠한 내용의 개발을 수행할 지 그 범위와 개발 내용을 기술할 것.

가. 개발 범위

타이머 기능을 담고 있는 module 형태의 디바이스 드라이버를 개발한다. 이때, 타이머는 커널의 타이머 작동을 이용하며, jiffies 를 활용한다. 또한 fpga 출력은 기존의 fpga device driver 코드를 참조하여 출력할 수 있도록 개발한다.

해당 device driver 를 실제로 구동할 수 있는 유저 프로그램을 작성한다. 구동하는 방법은 ioctl 을 사용하는 것으로 한다.

나. 개발 내용

디바이스 드라이버는 커널 모듈 형태로 개발한다. file_operation 을 지정하여 user program 에서 사용할 함수명과 실제로 드라이버에서 실행될 함수를 지정한다. Insmod 시 실행될 함수에서 register 에 해당 드라이버를 저장하고, 조작할 fpga device 의 주소를 mapping 한다. 또한 timer 를 초기화 하도록 한다,

open 할 때에는 이미 해당 드라이버가 오픈되어 있는지 확인할 수 있도록 한다. User program 에서 ioctl 함수를 활용할 것이기 때문에 해당 부분을 처리할 함수도 만들어 준다. IOCTL_SET_OPTION, IOCTL_COMMAND 두 경우에 대해서 처리할 수 있도록 한다.

Fpga 에 출력할 수 있는 함수를 따로 만들어 주어 지정된 데이터를 출력할 수 있도록 하고 close 를 처리할 수 있는 함수도 구현한다.

Timer 의 경우는 초기 상태를 설정하고 다음 타이머 등록과 타이머 상태 갱신을 해주며, parameter 로 받은 값에 따라 작동할 수 있도록 한다.

유저 프로그램은 IOCTL_SET_OPTION 을 통해 디바이스 드라이버에 입력받은 parameter 를 넘기고 명세된 기능을 실행할 수 있도록 한다.

III. 추진 일정 및 개발 방법

- 자신들이 설정한 개발 목표를 달성하기 위한 개발 일정을 설정하고, 각 요소 문제를 해결하기 위해서 어떤 방법을 사용할 지 기술할 것.

가. 추진 일정

- 5월 10일-13일 : 유저 프로그램 작성, 모듈 프로그램 작성 시작
- 5월 13일-16일 : 디바이스 드라이버 작성, 테스트
- 5월 17일: 보고서 작성

나. 개발 방법

- 디바이스 드라이버 개발

```
state cur_state;

static struct file_operations device_driver_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = device_ioctl,
    .open = device_open,
    .release = device_release,
};
```

디바이스 드라이버에서 file_operation 구조체에 Open, ioctl, release 에 해당하는 함수포인터를 각 연산 함수로 지정하였다.

-state 구조체 선언

```
// 출력 저장할 state
typedef struct state{
    struct timer_list timer; //timer
    char text[2][17];        //text lcd 저장할 변수
    bool mv_right[2];        // 오른쪽으로 갈 수 있는지 check
    short text_index[2];     //text lcd 출력 시작할 index
    int count;               //시간 count
    int end_cnt;             //end 하는 count
    int interval;            //timer interval
    short digit;             //현재 숫자 값
    short index;             // fnd 에 출력 하는 index
} state;
```

출력 상태를 저장할 state 구조체를 선언하였다.

구조체 멤버로는 timer, text lcd 값 저장 변수, text lcd 진행 방향 여부, text lcd 출력 시작 index, 시간 count , 시간 end count, interval, digit 변수, digit 을 fnd 에 출력할 위치 변수가 있다.

-module init 함수

```
3 //insmod 시 실행
9 static int __i const int registration
9 const int registration = register_chrdev(MAJOR_NUM, DEVICE_FILE_NAME, &device_driver_fops);
1
2
3 printk("device_init\n");
4
5 if (registration != 0)
5 return registration;
7
8
9 printk( "device file: /dev/%s ", DEVICE_FILE_NAME);
9 printk( "device major number: %d\n",MAJOR_NUM);
1
2
3 //fpga device driver mapping
4 iom_fpga_led_addr = ioremap(IOM_LED_ADDRESS, 0x1);
5 if(iom_fpga_led_addr == NULL)
5 printk("Failed to IO-map device %s\n" ,iom_fpga_led_addr);
7 iom_fpga_text_lcd_addr = ioremap(IOM_FPGA_TEXT_LCD_ADDRESS, 0x32);
8 if(iom_fpga_text_lcd_addr == NULL)
9 printk( "Failed to IO-map device %s\n" ,iom_fpga_text_lcd_addr);
9 iom_fpga_dot_addr = ioremap(IOM_FPGA_DOT_ADDRESS, 0x10);
1 if(iom_fpga_dot_addr == NULL)
1 printk( "Failed to IO-map device %s\n" ,iom_fpga_dot_addr);
3 iom_fpga_fnd_addr = ioremap(IOM_FND_ADDRESS, 0x4);
4 if(iom_fpga_fnd_addr == NULL)
5 printk( "Failed to IO-map device %s\n" ,iom_fpga_dot_addr);
5
6
7 //timer init
8 init_timer(&(cur_state.timer));
9
9 return 1;
1
}
```

Module init 함수는 다음과 같다.

해당 함수에서는 register_chrdev 에 major number, device file name 을 저장할 수 있도록 하며, 사용할 fpga hw 에 대하여 ioremap 을 진행하여 사용할 주소 값을 저장한다.
또한 init_timer 함수로 timer 를 초기화 하는 작업도 진행한다.

-device open 함수

```
//open 함수시 실행
static int device_open(struct inode* inode, struct file* file) {
    printk("device open\n");

    if (driver_port_usage != 0) {
        return -EBUSY;
    }
    //이미 open 되어 있는지 확인
    driver_port_usage = 1;

    return 0;
}
```

위 코드는 device open 시 실행되는 함수로, 단순히 device 가 미리 오픈되어 있는지만 확인한다.

-device release 함수

```
//close 시 실행
static int device_release(struct inode* inode, struct file* file) {
    driver_port_usage = 0;
    return 0;
}
```

위 코드는 device close 시 실행되는 함수이다.

-ioctl 처리 함수

```


//ioctl시 실행
static long device_ioctl(struct file* file, unsigned int ioctl_num, unsigned long ioctl_param) {
    char buffer[11], temp[5] = {'\0'};
    char* param;
    long timerInterval=0, timerCount=0;
    int i, mul= 1;
    switch (ioctl_num) {
    case IOCTL_SET_OPTION: // ioctl option IOCTL_SET_OPTION의 경우
        printk("IOCTL_SET_OPTION\n");

        //param 가져오기
        param = (char *)ioctl_param;
        if (strncpy_from_user(buffer, param, strlen_user(param)) < 0)
        {
            return -1;
        }

        // timerInterval parsing
        strncpy(temp, buffer, 3);
        //printk("%s\n", temp);
        for (i = 2; i >= 0; i--)
        {
            timerInterval += ((temp[i] - '0') * mul);
            mul *= 10;
        }
        //timeInterval 값이 0이면 Error
        if (timerInterval == 0)
            return -1;

        // timerCount parsing
        strncpy(temp, buffer + 3, 3);

```

 Makefile에 대한 권장되는 확장을

```

        //printk("%s\n", temp);
        mul = 1;
        for (i = 2; i >= 0; i--)
        {
            timerCount += ((temp[i] - '0') * mul);
            mul *= 10;
        }
        if (timerCount == 0)
            return -1;

        strncpy(temp, buffer + 6, 4);
        printk("%s\n", temp);

        init_timer_state(temp, timerInterval, timerCount);
        //state initialize
        break;

    case IOCTL_COMMAND: // ioctl option IOCTL_COMMAND 경우

        printk("IOCTL_COMMAND\n");
        start_timer();
        break;

    }

    return 0;
}

```

위 코드는 유저 프로그램에서 ioctl 로 호출했을 때 실행되는 함수이다.

Switch 문을 활용하여 넘겨받은 ioctl_num 명령의 내용에 따라 IOCTL_SET_OPTION, IOCTL_COMMAND 두 경우로 나누어 함수가 처리될 수 있도록 한다.

IOCTL_SET_OPTION 의 경우 char * 타입의 param 을 parsing 하여 유저프로그램에 입력한 parameter 들을 각 변수에 저장할 수 있도록 한다.

그리고 저장한 변수에 따라 이전에 선언한 state 구조체의 초기값을 만들어 준다.

IOCTL_COMMAND 의 경우는 저장된 parameter 에 따라 timer 가 작동할 수 있도록 처리한다.

-state 구조체를 초기화 하는 함수

```
//상태 초기화
void init_timer_state(const char* init, const int interval, const int count)
{
    printk("initialize state !\n");
    int i;
    for (i = 0; i < 4; i++) {
        if (init[i] != '0') {
            cur_state.digit = init[i] - '0';
            cur_state.index = i;
            //digit이 뭔지, 해당 index가 어디서 시작하는지 저장
            break;
        }
    }

    cur_state.mv_right[0] = cur_state.mv_right[1] = true;
    // text lcd 가 오른쪽으로 감지 왼쪽으로 감지 저장

    cur_state.text_index[0] = 0;
    cur_state.text_index[1] = 0;
    //text lcd 시작 index 저장
    cur_state.count = 0;
    //time count 저장
    cur_state.end_cnt = count;
    //time end count 저장
    cur_state.interval = interval;
    //time interval 저장

    for(i=0; i<16; i++)
    {
        cur_state.text[0][i] = ' ';
        //text에 ' ' 문자로 모두 저장.
    }
}
```

```

    cur_state.text[0][i] = '\0';

    for(i = 0; i < st_id_size; i++)
    {
        cur_state.text[0][i] = st_id[i];
        //학번 저장
    }

    for(i=0; i<16; i++)
    {
        cur_state.text[1][i] = ' ';
        //text에 ' '문자로 모두 저장
    }
    cur_state.text[1][i] = '\0';
    for(i = 0; i < name_size; i++)
    {
        cur_state.text[1][i] = name[i];
        //이름 저장
    }

    // printk("initialize state !%d %d %d %d !%s\n",
    // cur_state.interval, cur_state.text[0], cur_
}

```

위 코드는 이전에 선언한 state 를 초기화 하는 함수이다. 해당 함수는 이전에 기술한 IOCTL_SET_OPTION 처리 부분에서 실행되는 함수이다.

출력할 digit 의 값, fnd 에서 출력하는 위치, text lcd 값, text lcd 출력 위치, 움직이는 방향, time count 등의 변수를 초기화 한다.

-timer 실행 함수

```

//start timer 함수
void start_timer(void) {
    del_timer_sync(&(cur_state.timer));

    printk("start timer!\n");

    cur_state.timer.expires = get_jiffies_64() + cur_state.interval * (HZ / 10);
    cur_state.timer.data = (unsigned long) &cur_state;
    cur_state.timer.function = timer_func;

    add_timer(&(cur_state.timer));
}

```

해당 함수는 timer 를 실행하는 함수로 jiffies 를 활용하여 타이머가 작동될 수 있도록 한다.

기존에 등록된 타이머가 있다면 이를 삭제하고 다음 타이머를 커널에 등록시킨다. 여기서 유저 프로그램에서 interval 단위를 0.1초로 하였으므로 $hz/10 * interval$ 로 time interval을 나타낼 수

있도록 하고, 해당 data, function 또한 저장하여 주어 expire 시 해당 function 을 해당 data 로 실행될 수 있도록 한다.

-timer_func 함수

```
//timer function
static void timer_func(unsigned long timeout) {

    state* next_state = (state*) timeout;

    printk("start update!\n");

    int i;
    next_state->count++;
    if (next_state->count == next_state->end_cnt) {

        next_state->digit = 0;
        memset(next_state->text[0], ' ', 16);
        memset(next_state->text[1], ' ', 16);
    }
    else {

        //digit update
        next_state->digit = (next_state->digit == DIGIT_END ? 1 : (next_state->digit + 1));
        //digit 값 증가
        if (next_state->count % DIGIT_END == 0){
            //바뀐 count가 8이 되면 출력 index 증가
            next_state->index = (next_state->index + 1) % 4;
        }

        // text lcd update
        for(i =0;i<BUF_SIZE;i++){
            next_state->text[0][i] = ' ';
            next_state->text[1][i] = ' ';
        }
    }
}
```

Makefile에 대한 권

```

if(next_state->mv_right[0])
{
    //오른쪽으로 가는 경우
    next_state->text_index[0] += 1;
    for(i = 0;i<st_id_size;i++)
        next_state->text[0][next_state->text_index[0] + i] = st_id[i];
    if(next_state->text_index[0] + st_id_size == BUF_SIZE)// 더 이상 못가면 다음부터는 왼쪽으로
        next_state->mv_right[0] = false;
}
else{
    //왼쪽으로 가는 경우
    next_state->text_index[0] -= 1;
    for(i = 0;i<st_id_size;i++)//학번 저장
        next_state->text[0][next_state->text_index[0] + i] = st_id[i];
    if(next_state->text_index[0] == 0)//더 이상 못가면 다음부터는 오른쪽으로 갈 수 있게
        next_state->mv_right[0] = true;
}
if(next_state->mv_right[1])
{
    //오른쪽으로 가는 경우
    next_state->text_index[1] += 1;
    for(i = 0;i<name_size;i++)//이름 저장
        next_state->text[1][next_state->text_index[1] + i] = name[i];
    if(next_state->text_index[1] + name_size == BUF_SIZE)// 더 이상 못가면 왼쪽으로 가도록
        next_state->mv_right[1] = false;
}
else{
    //왼쪽으로 가는 경우
    next_state->text_index[1] -= 1;

    for(i = 0;i<name_size;i++) // 이름 저장
        next_state->text[1][next_state->text_index[1] + i] = name[i];
    if(next_state->text_index[1] == 0)// 더 이상 못가면 오른쪽으로

```

```

else{
    //왼쪽으로 가는 경우
    next_state->text_index[1] -= 1;

    for(i = 0;i<name_size;i++) // 이름 저장
        next_state->text[1][next_state->text_index[1] + i] = name[i];
    if(next_state->text_index[1] == 0)// 더 이상 못가면 오른쪽으로
        next_state->mv_right[1] = true;
}

// Register next timer
next_state->timer.expires = get_jiffies_64() + next_state->interval * (HZ / 10);
next_state->timer.data = (unsigned long) &cur_state;
next_state->timer.function = timer_func;
add_timer(&(amp;next_state->timer));
}
printk("finish update!\n");
fpga_write();
printk("write update!\n");
// Print updated state
//print_state(payload);
}

```

State 를 update 하는 함수이다. Digit 값 변경 update, digit 출력 index update, text lcd 출력 index update, 출력 방향 update , timer update 가 이루어 진다.

Count 가 end count 까지 이루어 지면, 모든 값은 0 으로 초기화 하고 종료할 수 있도록 한다.

-fpga write 함수

```
//fpga write 하는 함수
void fpga_write(void)
{
    //dot_write
    int i;

    unsigned char *value;
    unsigned short int _s_value;

    value = fpga_number[cur_state.digit];
    for(i=0;i<10;i++)
    {
        outw(value[i] & 0x7F,(unsigned int)iom_fpga_dot_addr+i*2);
    }

    //fnd write

    unsigned short int value_short = 0;

    value_short = cur_state.digit << (12 - 4 * cur_state.index);
    outw(value_short,(unsigned int)iom_fpga_fnd_addr);

    // led write
    const unsigned short led_value = (1 << (8 - cur_state.digit));
    outw(led_value, (unsigned int)iom_fpga_led_addr);

    //text_lcd write

    unsigned char text_value[33];
    text_value[32] = 0;
```

```
//text_lcd write

    unsigned char text_value[33];
    text_value[32] = 0;

    for(i = 0;i<16;i++)
    {
        text_value[i] = cur_state.text[0][i];
        text_value[16+i] = cur_state.text[1][i];
    }
    printk("!!%s!\n",text_value);
    for(i=0;i<32;i++)
    {
        outw((text_value[i] & 0xFF) << 8 | text_value[i + 1] & 0xFF,(unsigned int)iom_fpga_text_lcd_addr+i);
        i++;
    }
}
```

해당 함수는 state 구조체의 값을 토대로 fpga 보드에 알맞게 출력할 수 있도록 하는 함수이다.
Dot 출력, fnd 출력, led 출력, text lcd 출력에 각각에 맞게 데이터를 수정하여 outw 함수를
이용하여 출력될 수 있도록 한다,

-user program

```
int main(int argc, char* argv[]) {  
    int interval;  
    int count;  
    int init;  
    int i, hasNonZero;  
  
    char data[15] = {'\0'};  
  
    // Argument 개수 check  
    if (argc != 4) {  
        printf("argument error!\n");  
        return -1;  
    }  
  
    //interval, count, init 변수 할당  
    interval = atoi(argv[1]);  
    count = atoi(argv[2]);  
    init = atoi(argv[3]);  
  
    //만약 값이 0 이면 error!  
    if (interval == 0 || count == 0)  
    {  
        printf("Error parsing arguments!\n");  
        return -1;  
    }  
  
    // interval boundary check  
    if (interval < 1 || interval > 100)  
    {  
        printf("time interval value error!\n");  
        return -1;  
    }  
}
```

```

// init 값 check
int flag = 0;
if (strlen(argv[3]) != 4) {
    printf(" timer init length error!\n");
    return -1;
}
for (i = 0; i < 4; i++) {
    //값의 boundary 0~8 인지 check
    if ((argv[3][i] - '0') < 0 || (argv[3][i] - '0') > 8) {
        printf(" timer init value error!(0~8)\n");
        return -1;
    }
    if (argv[3][i] != '0') {
        //0이 아닌 값이 두개 이상인지 check
        if (flag != 0) {
            printf("timer init too many non-zero digit.\n");
            return -1;
        }
        flag = 1;
    }
}
if (flag == 0) {
    printf("timer init no non-zero digit.\n");
    return -1;
}

```

```

int fd = open(DEVICE_PATH, O_WRONLY);
//device driver check

if (fd == -1) {
    printf("file open error\n");
    return -1;
}

sprintf(data, "%03d%03d%04d", interval, count, init);

//ioctl로 전송
ioctl(fd, IOCTL_SET_OPTION, data);
ioctl(fd, IOCTL_COMMAND);

close(fd);

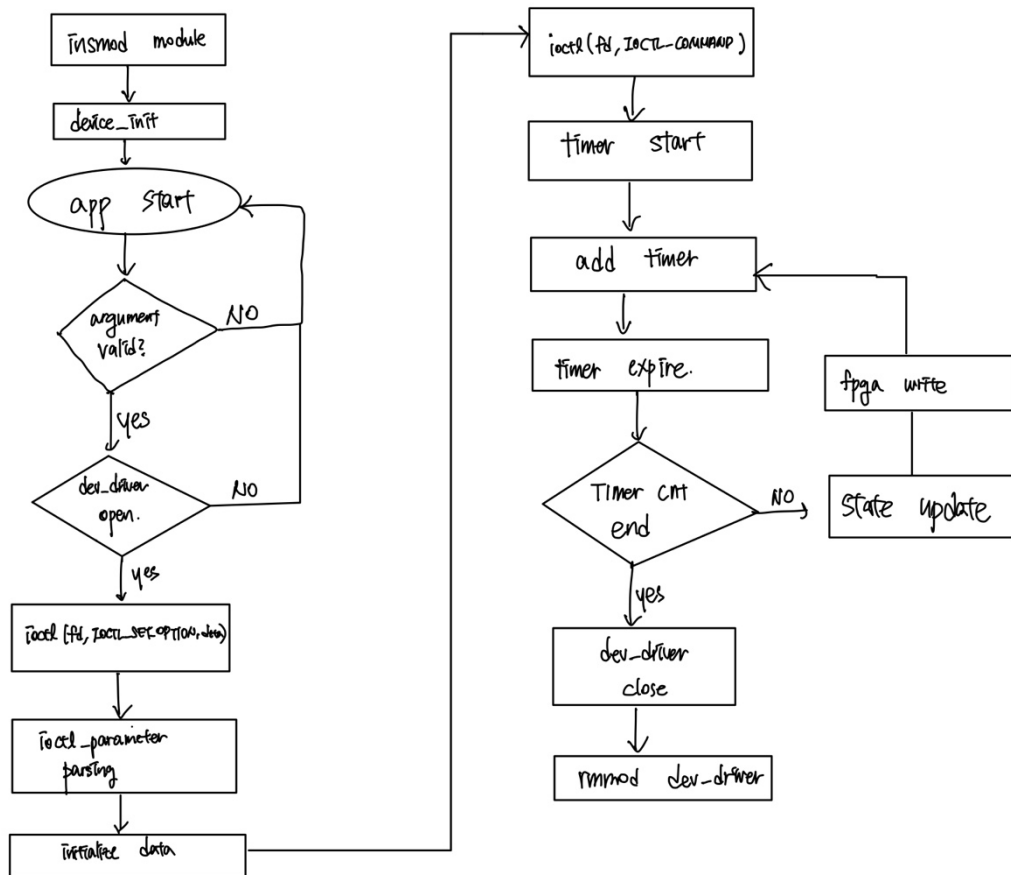
return 0;
}

```

유저 프로그램은 입력받은 argument 를 각각 변수에 따로 저장하여 조건에 맞는지 check 한다. 이후 open 함수를 이용하여 device driver 를 open 하고, ioctl 함수를 이용하여 argument 들을 data 변수에 저장 및 IOCTL_SET_OPTION 매크로를 전송하여 디바이스 드라이버에서 해당 함수 처리를 할 수 있도록 하고, IOCTL_COMMAND 매크로를 전송하여 타이머가 시작될 수 있도록 하였다.

모든 작업이 끝나면 close 함수를 호출하여 디바이스 드라이버를 close 할 수 있도록 한다.

최종 flow chart 는 다음과 같다.



IV. 연구 결과

- 최종 연구 개발 결과를 자유롭게 기술할 것.

명세서에 기술된 사항을 모두 충족 시키는 것을 확인할 수 있었다. 유저 응용 프로그램의 argument 의 값을 바꾸어 실행했을 때도, 해당 argument 값에 맞게 실행이 되었고,

Fpga_fnd, fpga_led, fpga_dot, fpga_text_lcd 요구하는 대로 출력이 되는 것을 확인할 수 있었다.

다만, 학번과 이름은 argument 로 입력받지 않으므로, 학번과 이름에 해당하는 값은 고정시켜 놓았기 때문에, 값을 변경하게 되면, 이름의 경우 길이가 달라질 수 있어 실행 시 오류가 날 수 있음을 확인할 수 있었다.

만약 text_lcd 에 해당하는 값을 바꾸게 되면 선언한 매크로의 값도 변경해야 함을 확인할 수 있다.

V. 기타

- 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

이번 과제 수행을 통해 디바이스 드라이버 작성 요령을 많이 배울 수 있게 되었고, 강의에서 배운 내용인 ioctl 관해서도 이론적으로만 이해하는 것이 아닌 실제로 어떻게 사용이 되는지 또한 배울 수 있어 많은 도움이 되었던 과제 였습니다.

또한 fpga 작동 드라이버도 그냥 작성된 드라이버를 쓰는 것이 아니라 직접 드라이버 코드를 분석해가며, 어떻게 fpga 작동을 시키는 코드를 작성하는지 알아낼 수 있었고, 이를 타이머 디바이스 드라이버에 적용을 시키는 경험을 할 수 있었다.

해당 과제에서는 button 입력과 관련된 기능은 없었기 때문에 해당 기능을 추가해보고 싶다는 생각도 해보았다. 예를들어 학번과 이름을 직접 입력할 수 있도록 하는 것도 재미있을 것 같다는 생각을 했다.

디바이스 드라이버를 직접 작성해보며, 유저 프로그램과 모듈, 하드웨어의 흐름이 어떻게 이루어지는지 보다 더 직접 많은 것을 알 수 있었던 좋은 기회가 되었던 것 같다.