Embedded System Software 과제 3

(과제 수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20181625, 남현준

개발기간: 2024. 05. 20. -2024. 06. 04.

최 종 보 고 서

I. 개발 목표

- 각 과제마다 주어지는 주제를 바탕으로 본 과제에서 추구하는 개발 목표를 설정하고 그 내용을 기술할 것.

Top & bottom half 방식으로 진행되는 interrupt handling의 흐름을 이해하고, 이해한 내용을 바탕으로 top half, bottom half로 나눠서 interrupt handling을 처리하는 모듈의 구현을 진행한다. 해당 과제에서는 stopwatch 기능의 구현을 진행하며 디바이스 드라이버를 통해 수행될 수 있도록 한다. FPGA의 HOME, BACK, VOL+, VOL- 버튼을 통해 특정 interrupt를 발생시키고 명세서의 내용대로 각 버튼에 맞는 interrupt handling이 진행될 수 있도록 interrupt handler를 구현한다. 이 때, stopwatch의 분, 초, 밀리초 단위의 내용이 FPGA FND, DOT 상에 그려지도록 적절하게 module을 제어할 수 있도록 한다.

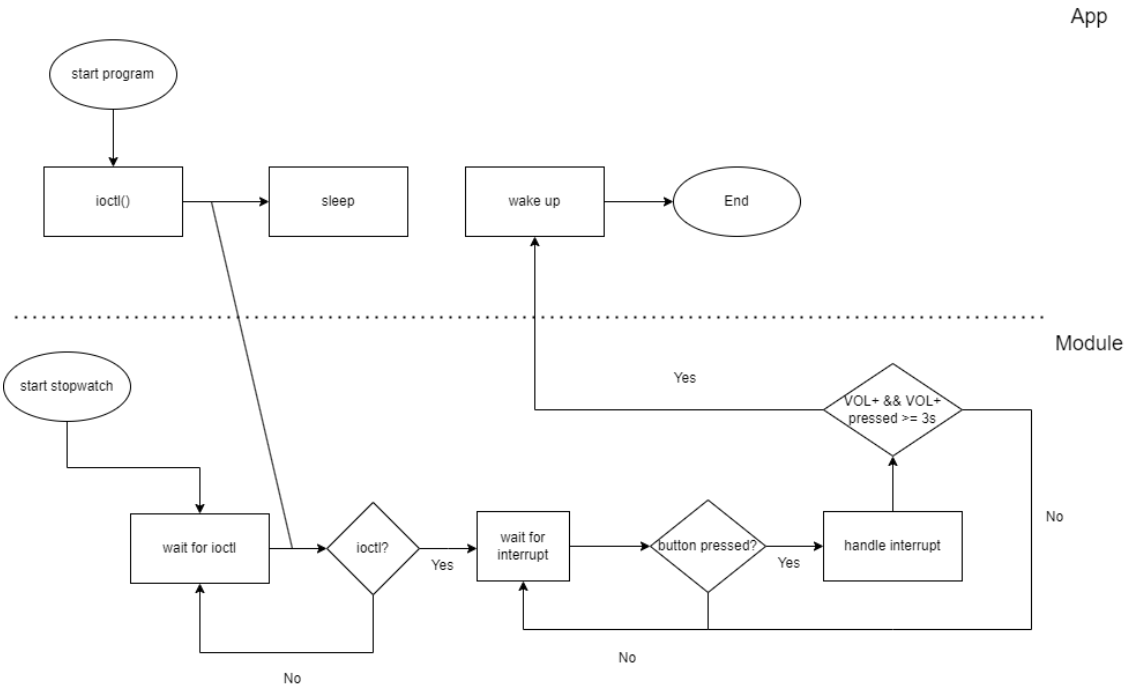
II. 개발 범위 및 내용

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

가. 개발 범위

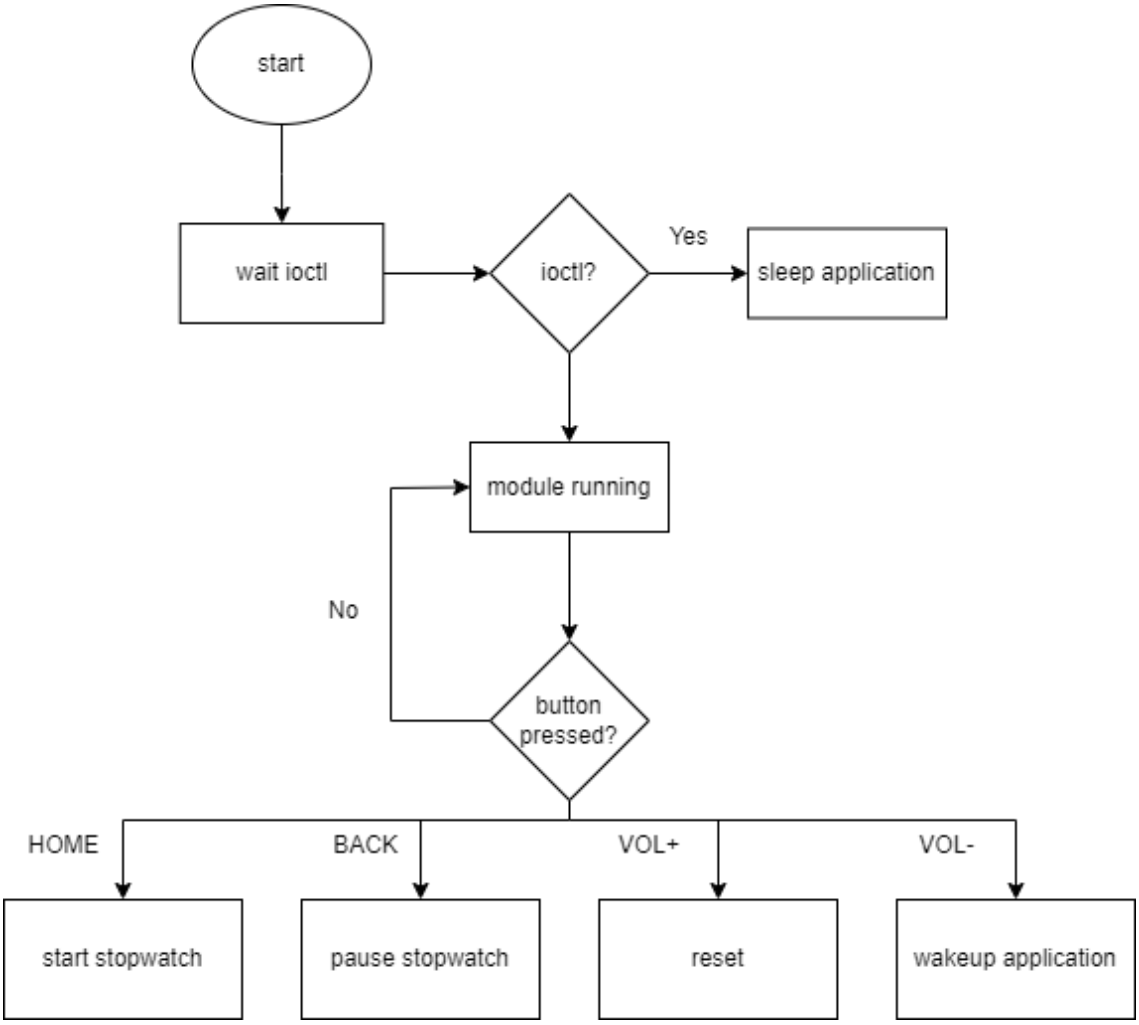
- ♦ Application

  Stopwatch 디바이스에 명령을 전달하게 되는 응용 프로그램의 개발을 진행한다. 이 때, 응용 프로그램에서 ioctl()을 통해 적절하게 device driver에서 명령을 처리할 수 있도록 구현을 진행한다.



♦ Device driver

Stopwatch 디바이스의 디바이스 드라이버 구현. Ioctl()을 통해 구동 명령이 주어지는 경우, 응용 프로그램을 stopwatch 디바이스가 종료되는 시점까지 sleep 상태로 전환되도록 한다. 명세서의 spec에 맞게 FPGA 버튼 입력에 대응하는 interrupt를 설치하고, 각 interrupt 발생 시에 이를 처리하는 handler를 top/bottom half로 나눠서 time-sensitive한 작동은 top half, 그 외 나머지 부분은 bottom-half에서 처리될 수 있도록 구현한다.



나. 개발 내용

♦ 응용 프로그램

응용 프로그램에서 stopwatch 디바이스에 디바이스 드라이버를 통해 ioctl() 명령을 전달할 수 있도록 open(), close(), ioctl() 등을 정의하고 구현한다. 이후, ioctl()을 통해 stopwatch 디바이스에 구동 명령이 주어지는 경우, 디바이스에서의 작동이 끝나 wake up되기 전까지 sleep 상태로 유지된다.

♦ 디바이스 드라이버

다음과 같은 버튼 입력에 대해 interrupt handling이 수행될 수 있도록 한다

- HOME
- BACK
- VOL+

- VOL-

과제 명세서의 내용에 맞게 다음과 같은 조건을 만족하도록 구현한다:

- /dev/stopwatch, major number 242로 구현을 진행한다
- 응용 프로그램이 시작 후, 디바이스에서 wake up이 되기 전까지 sleep 상태로 유지된다.
- 응용 프로그램이 실행 중 상태라면 FND의 초기 상태는 0000으로 설정한다.
- 스톱워치의 구현에서 각 버튼의 interrupt는 top/bottom half로 나눠서 구현이 진행되며 top-half에서 time-sensitive한 작동의 처리, bottom-half에서 top-half에서 완료되지 못한 나머지 작업들의 처리를 진행할 수 있도록 한다.
- HOME 버튼으로 스톱워치의 작동을 시작한다.
- 스톱워치가 작동 중인 경우, 0.1초마다 DOT의 값(=밀리초)을 갱신하고 FND 상의 4 length string 중에서 0, 1번째 idx로 분(= minute), 2, 3번째 idx로 초(= second)의 값이 출력될 수 있도록 한다.
- BACK 버튼을 누르는 경우, 스톱워치를 일시정지 시킨다. 일시정지 상태에서 다시 BACK을 누르는 경우, 스톱워치를 다시 실행시킨다. 이 때, 기존의 시간 (minute, second, millisecond)정보는 유지된 상태에서 갱신이 이루어져야 한다.
- VOL+ 버튼을 누르는 경우 reset이 일어나게 되며 스톱워치의 초기 상태로 되돌린다.
- VOL- 버튼을 3초 이상 누르는 것으로 디바이스의 종료 및 sleep 상태의 응용 프로그램을 wake up시켜 종료될 수 있도록 한다. 3초를 누르는 중간에 버튼에서 손을 떼는 경우, 다시 3초 이상 누르는 것을 유지해야 종료되어야 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

2024.05.20 – 2024.05.21 : 명세서 정독 및 프로그램 구조화

2024.05.21 – 2024.05.31 : 응용 프로그램, driver top/bottom half 구현

2024.06.01 – 2024.06.03 : 보고서 작성

나. 개발 방법

- 응용 프로그램

  Ioctl()을 통해 스톱워치 디바이스를 구동하라는 명령을 내린다.

```
int fd = open(DEVICE_PATH, O_WRONLY);
if (fd < 0) {
    printf("failed to open /dev/stopwatch₩n");
    return -1;
}

ioctl(fd, IOCTL_START);

//to check program entered sleep state, and is terminated when VOL+ is pressed for about 3 secs
printf("program terminated after sleep₩n");
close(fd);
```

  스톱워치 디바이스에서 VOL- 버튼이 3초 이상 눌러져 디바이스가 종료되며 wake up시키는 경우에만 응용 프로그램의 close(fd)가 호출되며 종료가 진행될 것이다.

- 디바이스 드라이버

```
static struct file_operations driver_fops = {
    .owner = THIS_MODULE,
    .open = driver_open,
    .release = driver_release,
    .unlocked_ioctl = driver_ioctl,
};
```

  디바이스의 작동을 명시하는 driver_fops를 정의한다.

```c
/*
    DATA data structure : data structure that will be used by implemented stopwatch.
*/
typedef struct _data{
    int minute;
    int second;
    int millisec;

    struct timer_list timer;
    struct timer_list exit_timer;

    long unsigned int prev_irq_time;
    int prev_irq;

    bool started;
    bool paused;
    bool quit;
    bool reset;
    bool timer_exit_executed;
}DATA;

DATA stopwatch;
```

해당 과제에서 스톱워치 디바이스에서 stopwatch와 연관된 정보를 저장하는 data structure를 선언한다.

```c
/*
    driver_init() : register device and map memories of FPGA modules
*/
static int __init driver_init(void) {
    printk("driver_init()₩n");
    int ret = register_chrdev(DEVICE_MAJOR_NUMBER, DEVICE_NAME, &driver_fops);
    if(ret != 0){
        printk("register error₩n");
        return ret;
    }

    /* Physical mapping for FND */
    fpga_fnd = ioremap(FPGA_FND_ADDR, 0x02);
    if (fpga_fnd == NULL) {
        printk("failed to ioremap fpga_fnd₩n");
        return -1;
    }
    fpga_dot = ioremap(FPGA_DOT_ADDR, 0x10);
    if(fpga_dot == NULL){
        printk(KERN_ERR "physical mapping unsuccessful.₩n");
        return -1;
    }
    //status_reset();
    //fpga_dot_clear();

    return 0;
}
```

드라이버를 설치하는 과정에서 initialization을 진행한다. 과제에서 사용할 스톱워치 디바이스와 해당 디바이스의 작동을 기반으로 character device의 등록을 진행한다. 또한, 스톱워치 디바이스에서 사용하는 FPGA 모듈인 FND와 DOT 모듈에 대해 ioremap()을 사용하여 physical memory와의 mapping을 진행하도록 한다.

```
if(already_opened) return -1;
already_opened = true;

/* Install GPIO handler */

/* HOME button */
gpio_direction_input(IMX_GPIO_NR(1, 11));
irqs[0] = gpio_to_irq(IMX_GPIO_NR(1, 11));
ret = request_irq(irqs[0], button_handler, IRQF_TRIGGER_RISING, "HOME", 0);

/* VOL+ button */
gpio_direction_input(IMX_GPIO_NR(2, 15));
irqs[1] = gpio_to_irq(IMX_GPIO_NR(2, 15));
ret = request_irq(irqs[1], button_handler, IRQF_TRIGGER_RISING, "VOLUP", 0);

/* VOL- button */
gpio_direction_input(IMX_GPIO_NR(5, 14));
irqs[2] = gpio_to_irq(IMX_GPIO_NR(5, 14));
ret = request_irq(irqs[2], button_handler, IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING, "VOLDOWN", 0);

/* BACK button */
gpio_direction_input(IMX_GPIO_NR(1, 12));
irqs[3] = gpio_to_irq(IMX_GPIO_NR(1, 12));
ret = request_irq(irqs[3], button_handler, IRQF_TRIGGER_RISING, "BACK", 0);

/* Install end */

status_reset();
```

```
static void status_reset(void){
    stopwatch.minute = 0;
    stopwatch.second = 0;
    stopwatch.millisec = 0;

    stopwatch.quit = false;
    stopwatch.started = false;
    stopwatch.paused = false;
    stopwatch.timer_exit_executed = false;

    char fnd[5] = "0000";
    fpga_fnd_init(fnd);
    fpga_dot_init(0);
}
```

디바이스를 open()하는 경우, 디바이스에서 사용할 interrupt 종류를 install하고 FND, DOT 등을 초기 상태인 0000, 0의 값으로 초기화를 진행한다.

Interrupt IRQ를 install하는 과정에서 trigger rising/falling으로 parameter를 주게 되는 HOME, BACK, VOL+의 경우 버튼이 눌러지는 경우 처리되어야 하므로

IRQF_TRIGGER_RISING 옵션을 주는 것으로 눌렀다 떼는 일련의 과정이 완료될 때 interrupt를 생성하도록 한다.

VOL-의 경우, RISING, FALLING 옵션을 둘 다 주는 것으로 버튼을 누르거나 떼는 경우 모두 interrupt를 발생시켜 총 시간이 3초 정도 되었는지 체크할 수 있도록 한다.

```
static long driver_ioctl(struct file* file, unsigned int cmd, unsigned long arg) {
    printk("driver_ioctl()₩n");

    switch(cmd){
        case IOCTL_START:
            wait_event_interruptible(wait_queue, stopwatch.timer_exit_executed == true);
            printk("sleep done₩n");
            break;
        default:
            return -1;
            break;
    }
    return 0;
}
```

ioctl()을 통해 명령이 주어지는 경우, 명령의 종류에 따라 ioctl을 처리할 수 있도록 한다. 해당 과제에서는, 한 종류의 ioctl()이 존재하며 해당 요청이 주어지는 경우 응용 프로그램을 sleep 상태로 대기시킨다.

```c
static irqreturn_t button_handler(int irq, void* dev_id) {
    unsigned long cur_press_time = get_jiffies_64();

    /*
        logic to avoid same irq to be requested in short period of time.
        if requested irq is other than VOLDOWN irq, then if interval between previous request and current request
        is less than 0.3 sec, do no handle.
    */
    if(irq == stopwatch.prev_irq && irq != irqs[3]){
        if(cur_press_time - stopwatch.prev_irq_time < (HZ / 10) * 3) return;
    }

    stopwatch.prev_irq = irq;
    stopwatch.prev_irq_time = get_jiffies_64();
    task_data.irq = irq;

    if(irq == irqs[0]){  //HOME
        printk("HOME KEY pressed\n");
        if(!stopwatch.started)
            stopwatch_set();
        //setFND("1111");
    }
    else if(irq == irqs[1]){  //VOL+
        printk("VOLUP pressed\n");
        stopwatch.reset = true;
    }
    else if(irq == irqs[2]){  //VOL-
        printk("VOLDOWN pressed\n");
        stopwatch.quit = !stopwatch.quit;

    }
    else{               //BACK
        if(stopwatch.started){
            printk("BACK pressed\n");
            stopwatch.paused = !stopwatch.paused;
        }
    }

    tasklet_schedule(&tasklet_proc);

    return IRQ_HANDLED;
}
```

```c
//MACRO for TASKLET execution, execute button_handler_bottom with using tasklet_proc as data
static DECLARE_TASKLET(tasklet_proc, button_handler_bottom, (unsigned long)&task_data);
```

Interrupt 발생 시, button handler가 호출되어 발생한 interrupt의 IRQ에 맞는 작동이 실행되도록 한다. 이 때, button handler는 top-half로 우선적으로 IRQ의 처리 여부만을 return하고 실질적인 handling은 TASKLET을 이용해 scheduling을 진행한 다음 bottom-half에서 처리되도록 한다.

```
static void button_handler_bottom(unsigned long data){
    TASKLET_DATA *tmp = (TASKLET_DATA *)data;

    if(tmp->irq == irqs[0]){ //HOME

    }
    else if(tmp->irq == irqs[1]){ //VOLUP, reset FPGA module contents, set stopwatch to initial state
        stopwatch.reset = false;
        status_reset();
        stopwatch.reset == true ? printk("1\n") : printk("0\n");
        del_timer(&stopwatch.timer);
    }
    else if(tmp->irq == irqs[2]){ //VOLDOWN, add exit timer. if it is keep pressed, update timer. otherwise, delete exit timer.
        if(stopwatch.quit){
            init_timer(&stopwatch.exit_timer);
            stopwatch.exit_timer.expires = get_jiffies_64() + HZ/10 * 30;
            stopwatch.exit_timer.function = exit_blink;
            add_timer(&stopwatch.exit_timer);
        }
        else{ //delete exit timer
            printk("delete exit timer\n");
            del_timer(&stopwatch.exit_timer);
        }
    }
    else{ //BACK,  if stopwatch is executed, pause/unpause stopwatch when BACK input is given
        if(stopwatch.started){
            stopwatch.paused == true ? del_timer(&stopwatch.timer) : stopwatch_set(); //if true,
        }
    }
}
```

Handler_bottom은 bottom-half의 구현으로 TASKLET에 저장된 task를 순차적으로 interrupt IRQ에 맞네 bottom-half의 logic으로 처리를 진행한다.

HOME button은 단순히 디바이스를 시작하라는 interrupt이기 때문에 bottom-half에서의 구현은 없다.

VOL- 버튼은 디바이스의 종료를 담당하는 interrupt이다. 해당 interrupt 발생 시, expire 시간을 현재 기준 +3초로 설정하여 exit 로직을 처리하고 종료될 수 있도록 exit timer의 설정을 진행하고 등록한다. 이 과정에서 중간에 버튼에서 손을 떼는 경우, exit timer가 유지될 필요가 없기에 del_timer()를 통해 기존에 등록한 exit timer를 timer list에서 삭제하도록 한다.

VOL+ 버튼은 디바이스의 초기화를 진행하도록 하는 interrupt이다. 해당 interrupt 발생 시 status_reset()을 호출해 초기 상태로 되돌리고 스톱워치 수행간 등록된 timer가 있다면 해당 timer의 정보를 timer list에서 삭제하도록 한다.

BACK 버튼은 스톱워치를 pause/unpause시키는 interrupt이다. Pause상태가 되는 경우, 기존에 등록되어 있던 스톱워치의 timer를 삭제하는 것으로 시간 정보를 유지하며 스톱워치 디바이스의 갱신을 멈추고, unpause 시키는 경우 새롭게 스톱워치의 timer를 생성 및 등록하여 매 interval마다 다비이스의 갱신이 진행되도록 한다.

```c
static void stopwatch_proceed(void){
    //printk("stopwatch proceed()\n");
    stopwatch.millisec += 1;

    if(stopwatch.millisec == 10){
        stopwatch.millisec -= 10;
        stopwatch.second += 1;

        if(stopwatch.second == 60){
            stopwatch.second -= 60;
            stopwatch.minute += 1;
        }
    }

    char fnd[5];
    sprintf(fnd, "%02d%02d", stopwatch.minute, stopwatch.second);
    fpga_fnd_init(fnd);
    fpga_dot_init(stopwatch.millisec);
}

/*
    stopwatch_blink() : function that will be executed when stopwatch timer expires.
                        update stopwatch information by using stopwatch_proceed().
*/
static void stopwatch_blink(unsigned long data){
    stopwatch_proceed();

    stopwatch.timer.expires = get_jiffies_64() + HZ / 10; //0.1 sec interval
    stopwatch.timer.function = stopwatch_blink;
    add_timer(&stopwatch.timer);
}
```

과제에서 사용하는 timer에는 stopwatch의 갱신에 사용하는 timer, exit 판별 과정에서 사용하는 timer가 존재한다.

스톱워치 디바이스는 0.1초 단위로 갱신이 진행되며 갱신 과정에서 스톱워치에 저장되는 시간 (minute, second, millisecond) 정보를 갱신하고, 그에 맞춰 FND, DOT에 출력되는 내용 또한 갱신될 수 있도록 한다.

```c
/*
    exit_blink() : function that will be executed when exit timer expires
                   wake up currently sleeping process.
*/
static void exit_blink(unsigned long data){
    printk("exit_blink()");
    del_timer(&stopwatch.timer);
    status_reset();
    fpga_dot_clear();
    __wake_up(&wait_queue, 1, 1, NULL);
    stopwatch.quit = false;
    stopwatch.timer_exit_executed = true;
}
```

Exit timer에서 timer가 expire되는 경우 실행되는 함수이다. 스톱워치가 실행 중인 경우 timer가 등록되어 있을 수가 있으므로 해당 timer의 삭제를 진행한다. 이후, 디바이스를 초기화하고 명세서의 내용에 맞게 DOT을 blank상태로 되돌린다. 이후, sleep 상태에 있던 응용 프로그램을 wake up시킨 후, 종료된다.

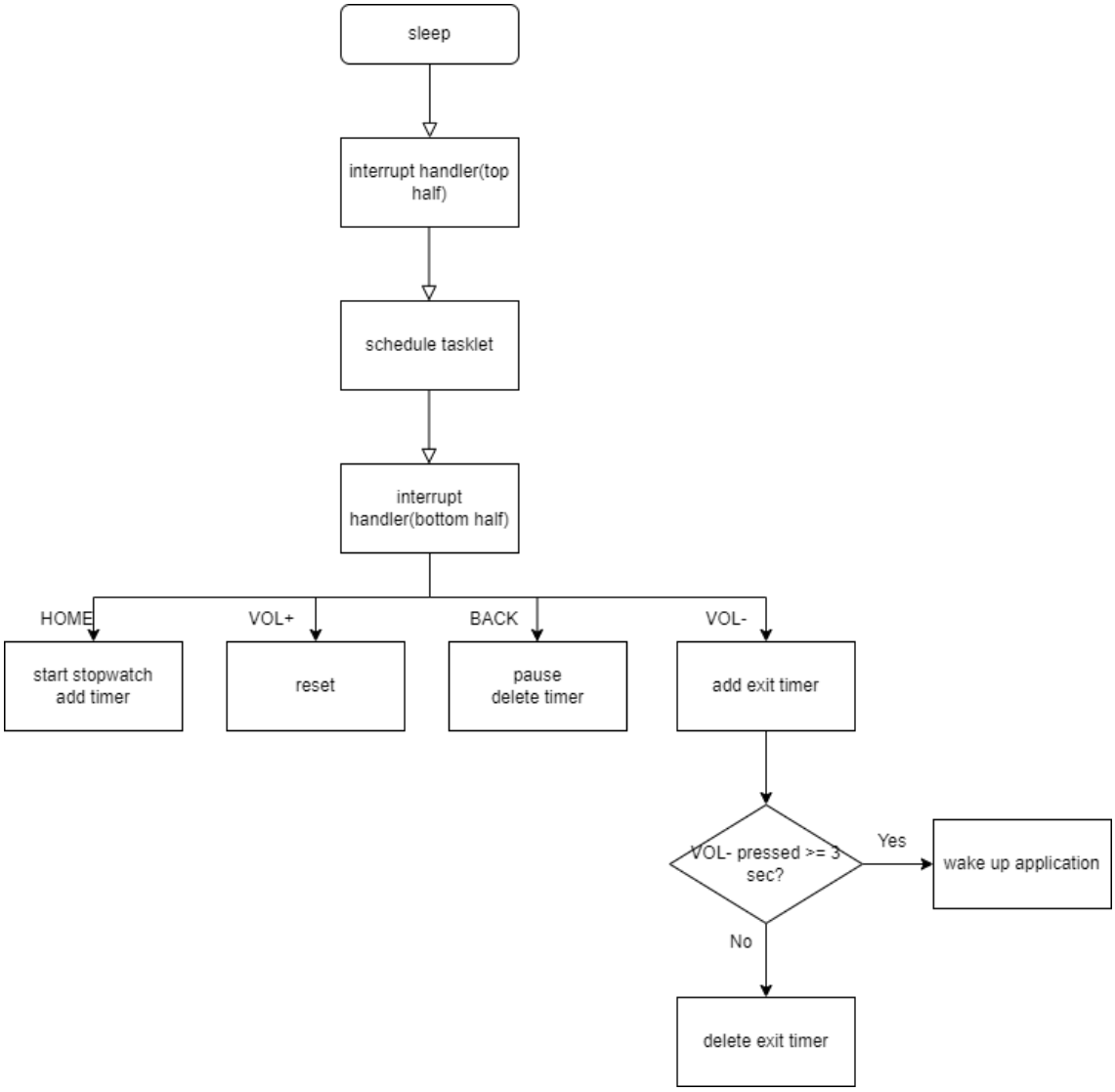
```
/*
    driver_release( ) : free installed IRQ, drop driver
*/
static int driver_release(struct inode* inode, struct file* file) {
    printk("driver_release( )₩n");
    already_opened = false;
    stopwatch.timer_exit_executed = false;
    free_irq(irqs[0], NULL);
    free_irq(irqs[1], NULL);
    free_irq(irqs[2], NULL);
    free_irq(irqs[3], NULL);
    return 0;
}
```

```
/*
    driver_exit( ) : unmap physical memories dedicated to FPGA modules and unregister device
*/
static void __exit driver_exit(void) {
    printk("driver_exit( )₩n");
    iounmap(fpga_fnd);
    iounmap(fpga_dot);
    unregister_chrdev(DEVICE_MAJOR_NUMBER, DEVICE_NAME);
}
```

종료 과정에서는 등록한 interrupt irq를 해제하고 디바이스에서 사용한 physical memory의 unmap을 진행하고 해당 device를 device 목록에서 삭제하도록 한다.

Ⅳ. 연구 결과

- 최종 연구 개발 결과를 자유롭게 기술할 것.



Interrupt handler의 top/bottom half의 flow는 다음과 같다. Top-half에서는 IRQ의 처리를 진행한 다음, tasklet의 scheduling을 통해 bottom-half에서 나머지 작업이 처리될 수 있도록 구현을 하게 된다.

이와 같이 top/bottom으로 handling하는 부분을 나눠서 interrupt handler를 구현해보는 것으로 device driver를 통해 장치 제어할 때 효율적으로 interrupt 처리를 할 수 있는 지식을 쌓고, 이를 구현하는데에 바탕이 되는 프로그래밍 능력을 키울 수 있었다.

V. 기타

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

과제를 구현하며 특정 함수는 module license의 등록 없이는 사용할 수 없었다.

이와 별개로, 과제 2까지는 일일이 필요한 .ko파일의 install/uninstall, 모듈의 install 등을 수동으로 진행했는데 주로 사용하는 명령어를 .sh파일에 적고 sh 파일이름.sh로 실행시키는 것으로 한 번에 필요한 동작을 수행시킬 수 있는 것이 편하다는 것을 이제야 알게 되었다.