Embedded System Software 과제 3

(과제 수행 결과 보고서)

과 목 명 : [CSE4116] 임베디드시스템소프트웨어

담당 교수 : 서강대학교 컴퓨터공학과 박성용

학번/이름 : 20151556 변홍수

개발 기간 : 2020. 05. 30. ~ 2020. 06. 01.

I. 개발 목표

Module Programming, 디바이스 드라이버, interrupt, timer를 활용하여 간단한 Stopwatch 프로그램을 작성한다

II. 개발 범위 및 내용

가. Application(테스트 응용 프로그램)

- ✓ Device파일 open()
- ✓ write()로 Stopwatch 시작하며 Sleep
- ✓ Wake Up(Stopwatch 종료)시, Device파일 close()후 프로그램 종료

나. Stopwatch Module

1) fpga_fnd(FND)

- ✓ 초기 상태는 '0000'이다.
- ✓ 앞 두 자리는 분(60분), 뒤 두 자리는 초(60초)를 표시한다.

2) GPIO Button

Button의 입력은 Interrupt를 이용하여 수행한다.

- ✓ Home : Start
  타이머를 시작한다. 1초마다 FND 출력을 갱신한다. **Kernel Timer를 사용한다.**
- ✓ Back : Pause
  타이머를 일시정지한다. **소수점 이하의 시간은 유지한다.**
- ✓ Volume Up : Reset
  초기 상태로 돌아간다.
- ✓ Volume Down : Exit
  3초 이상 누르고 있을 시 Application을 종료한다.
  FND를 '0000'으로 초기화한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

2020. 05. 30.

- 응용 프로그램 및 모듈 기초 구현
  - ✓ Volume Down버튼 3초 이상 눌렀을 시 기능 작동 테스트

2020. 05. 31.

- Stopwatch 구현
  - ✓ Kernel Timer 사용
  - ✓ 소수점 이하 유지
  - ✓ FND 작동
- GPIO Button 작동
  - ✓ Interrupt 사용
  - ✓ Home : Start, Back : Pause, Volume Up : Reset, Volume Down : Exit

2020. 06. 01.

- Document 작성

나. 개발 방법

1) Application(테스트 응용 프로그램)

✓ Device파일 open(), Open실패 시 프로그램 종료

✓ write()로 Stopwatch 시작하며 Sleep

✓ Wake Up(Stopwatch 종료)시, Device파일 close()후 프로그램 종료

app.c

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

```
int main(void){
	int fd;
	char buf;
	fd = open("/dev/stopwatch", O_RDWR);
	if (fd < 0){
		printf("Open Failured!\n");
		return -1;
	}
	write(fd, &buf, 1);
	close(fd);
	return 0;
}
```

Code 1

2) Stopwatch Module(dev_module.c)

- **Define Constant, Global Variable**

✓ unsigned long long start_time : Stopwatch를 시작할 때 jiffies값 저장

✓ int keep_time : Stopwatch가 Pause될 때 카운트 된 시간 값 저장

✓ bool timer_continue : Stopwatch가 실행 중일 때 true, 아닐 시 false 저장

```
#define DEVICE_MAJOR 242
#define DEVICE_NAME "stopwatch"
#define FND_ADDRESS 0x08000004 /
#define SUCCESS 0

/* Global Variable */
static int Device_Open = 0;
static unsigned char *fnd_addr;
struct timer_list timer;
struct timer_list end_timer;
bool timer_continue;
unsigned long long start_time;
int keep_time;

wait_queue_head_t wq_write;
DECLARE_WAIT_QUEUE_HEAD(wq_write);
```

Code 2

- **int stopwatch_open(struct inode *inode, struct file *file)**

✓ Device Open

✓ FND 초기화

✓ GPIO Button Interrupt Handler 등록

```
int stopwatch_open(struct inode *inode, struct file *file){
	int ret;
	int irq;
	printk(KERN_ALERT "Open Module\n");

	/* Don't want to run to two processes at the same time */
	if(Device_Open)
		return -EBUSY;
	Device_Open++;

	keep_time = 0;
	device_write_fnd(0);

	gpio_direction_input(IMX_GPIO_NR(1,11));
	irq = gpio_to_irq(IMX_GPIO_NR(1,11));
	ret=request_irq(irq, inter_handler1, IRQF_TRIGGER_FALLING, "home", 0);

	gpio_direction_input(IMX_GPIO_NR(1,12));
	irq = gpio_to_irq(IMX_GPIO_NR(1,12));
	ret=request_irq(irq, inter_handler2, IRQF_TRIGGER_FALLING, "back", 0);

	gpio_direction_input(IMX_GPIO_NR(2,15));
	irq = gpio_to_irq(IMX_GPIO_NR(2,15));
	ret=request_irq(irq, inter_handler3, IRQF_TRIGGER_FALLING, "volup", 0);

	gpio_direction_input(IMX_GPIO_NR(5,14));
	irq = gpio_to_irq(IMX_GPIO_NR(5,14));
	ret=request_irq(irq, inter_handler4,
			IRQF_TRIGGER_FALLING|IRQF_TRIGGER_RISING, "voldown", 0);

	return SUCCESS;
}
```

Code 3

- **int stopwatch_release(struct inode *inode, struct file *file)**

✓ Free Interrupt Handler

```
int stopwatch_release(struct inode *inode, struct file *file){
	/* Now Ready for our next caller */
	Device_Open--;

	free_irq(gpio_to_irq(IMX_GPIO_NR(1, 11)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(1, 12)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(2, 15)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(5, 14)), NULL);

	printk(KERN_ALERT "Release Module\n");
	return SUCCESS;
}
```

- **int stopwatch_write(struct file *filp, const char *buf,**
  size_t count, loff_t *f_pos

✓ User에서 write() 호출 시 wait queue에 추가한 뒤 Sleep한다.

```
int stopwatch_write(struct file *filp, const char *buf,
                    size_t count, loff_t *f_pos){
       printk("sleep on\n");
       interruptible_sleep_on(&wq_write);
       printk("write\n");
       return SUCCESS;
}
```

Code 4

- **int __init stopwatch_init(void)**

✓ Device Module을 등록한다.

✓ FND Physical Address를 Mapping한다

✓ Kernel Timer를 초기화한다.

```
int __init stopwatch_init(void) {
       int result;
       result = register_chrdev(DEVICE_MAJOR, DEVICE_NAME, &fops);
       if(result < 0) {
              printk(KERN_WARNING "Can't get any major\n");
              return result;
       }
       fnd_addr = ioremap(FND_ADDRESS, 0x4);
       init_timer(&timer);
       init_timer(&end_timer);
       printk(KERN_ALERT "Init Module Success\n");
       printk("%s major number : %d\n", DEVICE_NAME, DEVICE_MAJOR);

       return SUCCESS;
}
```

Code 5

- **void __exit stopwatch_exit(void)**

✓ Device Module을 제거한다.

✓ FND Physical Address를 UnMapping한다

✓ Kernel Timer를 삭제한다.

```
void __exit stopwatch_exit(void) {
	unregister_chrdev(DEVICE_MAJOR, DEVICE_NAME);
	iounmap(fnd_addr);
	del_timer_sync(&timer);
	del_timer_sync(&end_timer);
	printk(KERN_ALERT "Remove Module Success\n");
}
```

Code 6

- **void kernel_timer_write(unsigned long timeout)**

✓ bool timer_continue값이 false일 경우 함수를 즉시 종료한다.

✓ timer_time 값을 계산 한 뒤 device_write_fnd()를 호출해 FND을 출력한다.

✓ Kernel Timer의 expire, function값을 설정하고 add_timer()를 호출한다.

```
void kernel_timer_write(unsigned long timeout) {
	int timer_time;
	if(!timer_continue) return;
	timer_time = (get_jiffies_64() - start_time + keep_time);
	printk("raw data: %llu, expire time: %d, ",
		get_jiffies_64() - start_time + keep_time, HZ);
	device_write_fnd(timer_time);
	timer.expires = get_jiffies_64() + HZ;
	timer.function = kernel_timer_write;
	add_timer(&timer);
}
```

Code 7

- **void device_write_fnd(int timer_time)**

✓ timer_time값을 넘겨받아 분(int min), 초(int sec) 값을 계산하여 FND를 출력한다.

```
void device_write_fnd(int timer_time){
	int min, sec;
	unsigned char value[4];
	unsigned short int value_short = 0;
	timer_time /= HZ;
	sec = timer_time%60;
	timer_time /= 60;
	min = timer_time%60;
	printk("timer: %02d:%02d\n", min, sec);
	value[0] = min/10;
	value[1] = min%10;
	value[2] = sec/10;
	value[3] = sec%10;
	value_short = value[0] << 12 | value[1] << 8 | value[2] << 4 | value[3];
```

```
        outw(value_short, (unsigned int)fnd_addr);
}
```

Code 8

- **void stopwatch_end(unsigned long timeout)**

✓ FND를 초기화하고 __wake_up()을 호출하여 프로세스를 깨운다.

```
void stopwatch_end(unsigned long timeout){
        device_write_fnd(0);
        printk("Wake Up\n");
        __wake_up(&wq_write, 1, 1, NULL);
}
```

Code 9

- **irqreturn_t inter_handler1(int irq, void* dev_id)**

✓ bool timer_continue값이 true일 경우 함수를 즉시 종료한다. (이미 타이머 실행 중)

✓ **Pause상태에서 다시 Start된 경우 Start후 1초가 아니라 이전에 카운트 된 시간에서 정확히 그 다음 .00초가 되었을 때 FND출력을 갱신하기 위해, keep_time에 저장된 값을 사용하여 tmp_hz를 계산한다.**

✓ tmp_hz는 Kernel Timer Expire값에 사용된다.

```
irqreturn_t inter_handler1(int irq, void* dev_id) {
        int timer_time;
        int tmp_hz;

        if(timer_continue) {
                printk("Access denied : Timer is running.\n");
                return IRQ_HANDLED;
        }

        timer_continue = true;
        timer_time = keep_time;

        del_timer_sync(&timer);
        start_time = get_jiffies_64();

        tmp_hz = (int)keep_time % (int)HZ;
        tmp_hz = (int)HZ - tmp_hz;

        printk("Start timer.\n");
        printk("raw data: %llu, expire time: %d, ", keep_time, tmp_hz);
        device_write_fnd(timer_time);
```

```
        timer.expires = get_jiffies_64() + tmp_hz;
        timer.function = kernel_timer_write;
        add_timer(&timer);
        return IRQ_HANDLED;
}
```

Code 10

- **irqreturn_t inter_handler2(int irq, void* dev_id)**

✓ 타이머가 실행 중(timer_continue값이 true)일 때 타이머를 Pause한다.

✓ **다시 Start되었을 때 지금까지 저장된 시간을 유지하기 위해 지금까지 카운트 된 시간 값을 keep_time에 저장한다. kernel jiffies값의 차이를 사용하기 때문에 소수점 이하 둘째 자리(HZ=100)까지 표현할 수 있다.**

```
irqreturn_t inter_handler2(int irq, void* dev_id) {
        if(timer_continue){
                printk("Pause timer.\n");
                timer_continue = false;
                keep_time += (get_jiffies_64() - start_time);
        }
        return IRQ_HANDLED;
}
```

Code 11

- **irqreturn_t inter_handler3(int irq, void* dev_id)**

✓ bool timer_continue값이 true일 경우 함수를 즉시 종료한다. (이미 타이머 실행 중)

✓ keep_time과 FND출력을 초기화하여 Stopwatch Timer를 Reset한다.

```
irqreturn_t inter_handler3(int irq, void* dev_id) {
        if(timer_continue) {
                printk("Access denied : Timer is running.\n");
                return IRQ_HANDLED;
        }
        printk("Reset timer.\n");
        keep_time = 0;
        device_write_fnd(0);
        return IRQ_HANDLED;
}
```

Code 12

- **irqreturn_t inter_handler4(int irq, void* dev_id)**

✓ bool timer_continue값이 true일 경우 함수를 즉시 종료한다. (이미 타이머 실행 중)

✓ Volume Down 버튼이 눌렸을 시 end_timer를 등록한다. 타이머가 Expire되기 전에 버튼이 떼어졌을 경우 등록된 타이머를 삭제한다.

```
irqreturn_t inter_handler4(int irq, void* dev_id) {
	if(timer_continue) {
		printk("Access denied : Timer is running.\n");
		return IRQ_HANDLED;
	}
	if(gpio_get_value(IMX_GPIO_NR(5, 14))){
		printk("Keep timer.\n");
		del_timer(&end_timer);
	}
	else{
		printk("Exiting timer. Press the button 3seconds.\n");
		end_timer.expires = get_jiffies_64() + 3*HZ;
		end_timer.function = stopwatch_end;
		add_timer(&end_timer);
	}
	return IRQ_HANDLED;
}
```

Code 13

IV. 연구 결과

- fpga device driver와 interrupt를 포함한 stopwatch 기능을 가진 하나의 모듈 구현

- 프로그램 전체 흐름도

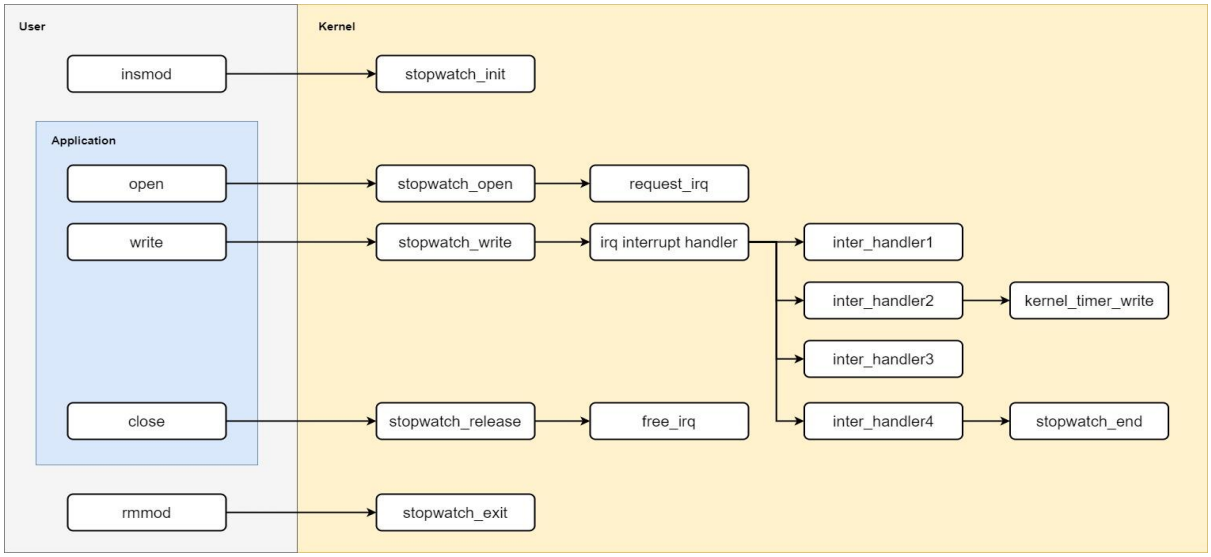

Figure 1

V. 기타

타이머를 오래 실행시켰을 때(실험적으로 6분 이상) 간헐적으로 printk()로 출력하는 jiffies값이 1이 증가하는 현상을 발견했다. 특정 케이스에서 반복해서 발견되는 현상이었으면 어느정도 납득을 할 수 있었겠지만 현상이 매우 불규칙적으로 발생하여 원인 파악과 해결을 할 수가 없었다. 커널 타이머와 jiffies관련 지식이 더 필요함을 느꼈다. 이번 과제가 마지막으로 리눅스 커널프로그래밍이 끝나 아쉽다.

I. Appendix

dev_module.c (entire)

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/interrupt.h>
#include <asm/io.h>
#include <mach/gpio.h>
#include <linux/timer.h>
#include <linux/wait.h>

#define DEVICE_MAJOR 242
#define DEVICE_NAME "stopwatch"
#define FND_ADDRESS 0x08000004
#define SUCCESS 0

/* Global Variable */
static int Device_Open = 0;
static unsigned char *fnd_addr;
struct timer_list timer;
struct timer_list end_timer;
bool timer_continue;
unsigned long long start_time;
int keep_time;

wait_queue_head_t wq_write;
DECLARE_WAIT_QUEUE_HEAD(wq_write);

/* Function Prototype */
int stopwatch_open(struct inode *, struct file *);
int stopwatch_release(struct inode *, struct file *);
int stopwatch_write(struct file *filp, const char *buf, size_t count, loff_t
*f_pos);
void kernel_timer_write(unsigned long timeout);
//void device_write_fnd(int min, int sec);
void device_write_fnd(int timer_time);
void stopwatch_end(unsigned long timeout);

irqreturn_t inter_handler1(int irq, void* dev_id);
irqreturn_t inter_handler2(int irq, void* dev_id);
irqreturn_t inter_handler3(int irq, void* dev_id);
```

```
irqreturn_t inter_handler4(int irq, void* dev_id);

/*
* Define file_operations structure.
* This structure will hold the functions to be called when a process does something
to the device we created.
* Since a pointer to this structure is kept in the devices table, it can't be
local to init_module.
* NULL is for unimplemented functions.
*/
static struct file_operations fops =
{
        .open = stopwatch_open,
        .write = stopwatch_write,
        .release = stopwatch_release,
};

irqreturn_t inter_handler1(int irq, void* dev_id) {
        int tmp_hz;

        if(timer_continue) {
                printk("Access denied : Timer is running.\n");
                return IRQ_HANDLED;
        }
        timer_continue = true;

        del_timer_sync(&timer);
        start_time = get_jiffies_64();

        tmp_hz = (int)keep_time % (int)HZ;
        tmp_hz = (int)HZ - tmp_hz;

        printk("Start timer.\n");
        printk("raw data: %d, expire time: %d, ", keep_time, tmp_hz);
        device_write_fnd(keep_time);

        timer.expires = get_jiffies_64() + tmp_hz;
        timer.function = kernel_timer_write;
        add_timer(&timer);
        return IRQ_HANDLED;
}

irqreturn_t inter_handler2(int irq, void* dev_id) {
        if(timer_continue){
                printk("Pause timer.\n");
                timer_continue = false;
                keep_time += (get_jiffies_64() - start_time);
        }
        return IRQ_HANDLED;
}

irqreturn_t inter_handler3(int irq, void* dev_id) {
        if(timer_continue) {
                printk("Access denied : Timer is running.\n");
                return IRQ_HANDLED;
        }
        printk("Reset timer.\n");
```

```
	keep_time = 0;
	device_write_fnd(0);
	return IRQ_HANDLED;
}

irqreturn_t inter_handler4(int irq, void* dev_id) {
	if(timer_continue) {
		printk("Access denied : Timer is running.\n");
		return IRQ_HANDLED;
	}
	if(gpio_get_value(IMX_GPIO_NR(5, 14))){
		printk("Keep timer.\n");
		del_timer(&end_timer);
	}
	else{
		printk("Exiting timer. Press the button 3seconds.\n");
		end_timer.expires = get_jiffies_64() + 3*HZ;
		end_timer.function = stopwatch_end;
		add_timer(&end_timer);
	}
	return IRQ_HANDLED;
}

int stopwatch_open(struct inode *inode, struct file *file){
	int ret;
	int irq;
	printk(KERN_ALERT "Open Module\n");

	/* Don't want to run to two processes at the same time */
	if(Device_Open)
		return -EBUSY;
	Device_Open++;

	keep_time = 0;
	device_write_fnd(0);

	gpio_direction_input(IMX_GPIO_NR(1,11));
	irq = gpio_to_irq(IMX_GPIO_NR(1,11));
	ret=request_irq(irq, inter_handler1, IRQF_TRIGGER_FALLING, "home", 0);

	gpio_direction_input(IMX_GPIO_NR(1,12));
	irq = gpio_to_irq(IMX_GPIO_NR(1,12));
	ret=request_irq(irq, inter_handler2, IRQF_TRIGGER_FALLING, "back", 0);

	gpio_direction_input(IMX_GPIO_NR(2,15));
	irq = gpio_to_irq(IMX_GPIO_NR(2,15));
	ret=request_irq(irq, inter_handler3, IRQF_TRIGGER_FALLING, "volup", 0);

	gpio_direction_input(IMX_GPIO_NR(5,14));
	irq = gpio_to_irq(IMX_GPIO_NR(5,14));
	ret=request_irq(irq,                                        inter_handler4,
IRQF_TRIGGER_FALLING|IRQF_TRIGGER_RISING, "voldown", 0);

	return SUCCESS;
}

int stopwatch_release(struct inode *inode, struct file *file){
```

```
	/* Now Ready for our next caller */
	Device_Open--;

	free_irq(gpio_to_irq(IMX_GPIO_NR(1, 11)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(1, 12)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(2, 15)), NULL);
	free_irq(gpio_to_irq(IMX_GPIO_NR(5, 14)), NULL);

	printk(KERN_ALERT "Release Module\n");
	return SUCCESS;
}

int stopwatch_write(struct file *filp, const char *buf, size_t count, loff_t
*f_pos){
	printk("sleep on\n");
	interruptible_sleep_on(&wq_write);
	printk("write\n");
	return SUCCESS;
}

/* Initialize the module – Register the character device */
int __init stopwatch_init(void) {
	int result;
	result = register_chrdev(DEVICE_MAJOR, DEVICE_NAME, &fops);
	if(result < 0) {
		printk(KERN_WARNING "Can't get any major\n");
		return result;
	}
	fnd_addr = ioremap(FND_ADDRESS, 0x4);
	init_timer(&timer);
	init_timer(&end_timer);
	printk(KERN_ALERT "Init Module Success\n");
	printk("%s major number : %d\n", DEVICE_NAME, DEVICE_MAJOR);

	return SUCCESS;
}

/* Exit the moudle - UnRegister the character device */
void __exit stopwatch_exit(void) {
	unregister_chrdev(DEVICE_MAJOR, DEVICE_NAME);
	iounmap(fnd_addr);
	del_timer_sync(&timer);
	del_timer_sync(&end_timer);
	printk(KERN_ALERT "Remove Module Success\n");
}

/* Write Timer */
void kernel_timer_write(unsigned long timeout) {
	int timer_time;
	if(!timer_continue) return;
	timer_time = (get_jiffies_64() - start_time + keep_time);
	printk("raw data: %llu, expire time: %d, ", get_jiffies_64() - start_time
+ keep_time, HZ);
	device_write_fnd(timer_time);
	timer.expires = get_jiffies_64() + HZ;
	timer.function = kernel_timer_write;
	add_timer(&timer);
```

```
}

void device_write_fnd(int timer_time){
	int min, sec;
	unsigned char value[4];
	unsigned short int value_short = 0;
	timer_time /= HZ;
	sec = timer_time%60;
	timer_time /= 60;
	min = timer_time%60;
	printk("timer: %02d:%02d\n", min, sec);
	value[0] = min/10;
	value[1] = min%10;
	value[2] = sec/10;
	value[3] = sec%10;
	value_short = value[0] << 12 | value[1] << 8 | value[2] << 4 | value[3];
    outw(value_short, (unsigned int)fnd_addr);
}

void stopwatch_end(unsigned long timeout){
	device_write_fnd(0);
	printk("Wake Up\n");
	__wake_up(&wq_write, 1, 1, NULL);
}

module_init(stopwatch_init);
module_exit(stopwatch_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Huins");
```

Code 14