Embedded System Software 과제 2

(과제 수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20181625, 남현준

개발기간: 2024. 05. 03. -2024. 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.

과제 2 에서는 수업에서 배운 내용을 바탕으로 실습에서 사용하는 Achorimx 보드의 FPGA 모듈을 활용해 타이머의 디바이스 드라이버 및 관련 테스트 응용 프로그램의 개발을 목표로 한다.

개발 과정에서, 응용 프로그램은 디바이스 드라이버를 기존의 방식이 아닌 ioctl 을 통해 타이머 디바이스의 제어를 하게 된다. Ioctl 을 통해 전달된 구동 옵션을 이용해 input 으로 주어지는 초기값을 기준으로 타이머의 초기화를 진행하고, 그 후 추가적인 ioctl 을 통해 타이머 디바이스가 명세서의 세부기능에 맞게 구동될 수 있도록 구현을 진행한다.

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

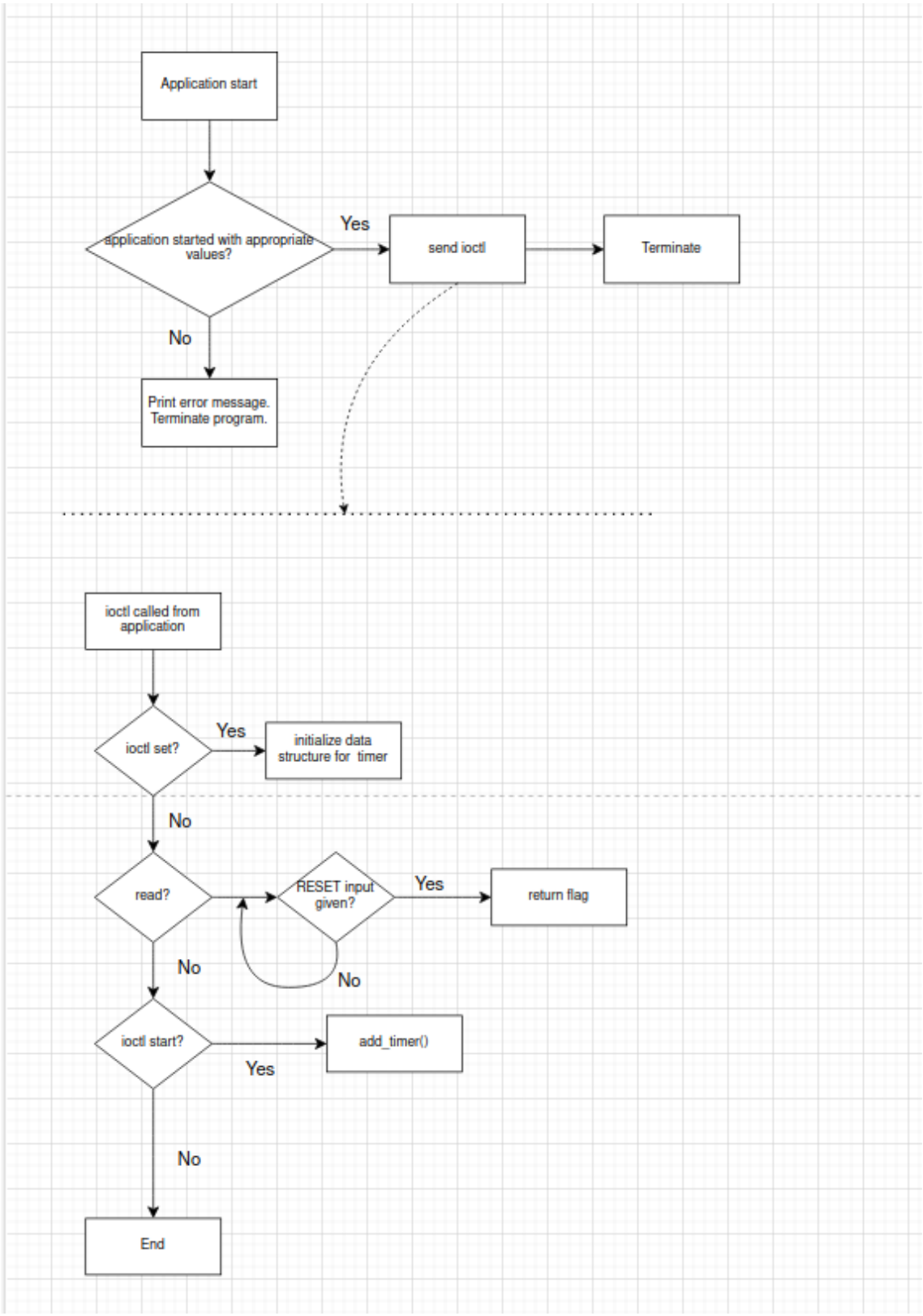
A. 응용 프로그램

타이머 디바이스의 구동에 사용될 초기값들을 입력받고, ioctl 을 통해 device driver 에 특정 요청을 전달하는 응용 application 의 개발을 진행한다. Ioctl 을 통해 전달되는 명령은 두 가지로, 초기값을 전달하여 타이머 디바이스의 initialization 을 진행하는 명령과 타이머 디바이스에 설정된 값을 이용한 실제 구동 명령을 내리는 ioctl 이 구현될 것이다.

B. 타이머 디바이스 드라이버

타이머 디바이스의 제어를 하게 되는 device driver 를 개발한다. 응용 프로그램으로부터 전달받는 값으로 임베디드 보드에 있는 FPGA 모듈들의 초기화를 진행하거나, 타이머 디바이스의 구동 명령을 받아 정의한 로직대로 타이머 디바이스가

구동될 수 있도록 한다.


Application start
application started with appropriate values?
Yes
send ioctl
Terminate
No
Print error message. Terminate program.
ioctl called from application
ioctl set?
Yes
initialize data structure for timer
No
read?
RESET input given?
Yes
return flag
No
No
ioctl start?
add_timer()
Yes
No
End

나. 개발 내용

A. 응용 프로그램

타이머 디바이스에 명령을 내리게 될 ioctl()와 관련된 개발을 진행한다. 과제 명세서에서 타이머 디바이스는 /dev/dev_driver 라는 이름을 가진다고 명시되어 있고, 이에 따라 /dev/dev_driver 에 대해 open(), ioctl(), read(), close()를 수행할 수 있도록 개발을 진행한다. 이 과정에서 ioctl 을 통해 전달되는 명령 중, 타이머 디바이스의 초기화와 관련된 부분에서 TIMER_INTERVAL 이 1 ~ 100, TIMER_CNT 가 1 ~ 200, TIMER_INIT 가 0001 ~ 8000 (입력 값이 00X0 과 같이 0 이 3 개, 1 ~ 8 범위의 숫자 1 개로 구성)의 범위를 가짐을 유의하고 해당 부분에 대해 예외 처리가 진행될 수 있도록 한다.

B. 타이머 디바이스

명령을 받게 되면, 타이머 디바이스는 다음과 같은 FPGA 모듈에 대해 제어를 하게 된다 :

1. DOT
2. FND
3. LCD
4. LED
5. PUSH_SWITCH

과제 명세서와 함께 주어진 과제 2_driver 의 내용을 참고하여 ioremap(), iounmap()과 같은 함수들을 이용해 FPGA 모듈을 memory mapped I/O 방식으로 활용될 수 있도록 physical address 에 매핑을 진행하고, outw() 함수를 통해 모듈에서 사용될 내용이 write 될 수 있도록 한다.

FPGA 모듈의 제어를 하는 과정에서 다음과 같이 작동될 수 있도록 한다.

- 사용하는 타이머 디바이스는 /dev/dev_driver, dev_driver.ko 를 통해 driver 의 설치, 제거 및 사용될 수 있도록 한다.
- 타이머 디바이스의 register 사용하는 device type 은 character device, Major number 는 242 로 설정하도록 한다.
- Ioctl 을 통해서 전달되는 초기값에서 TIMER_INTERVAL 은 1 ~ 200, TIMER_CNT 는 1 ~ 200, TIMER_INIT 은 0001 ~ 8000 사이의 범위에서만

전달될 수 있도록 하고 범위를 벗어나거나 조건에 어긋나는 경우 디바이스를 초기화하는 것이 아닌 에러를 출력하고 응용 프로그램이 종료될 수 있도록 한다.

- TIMER_INTERVAL 이 0 이 되기 전까지, 매 interval 마다 FPGA 모듈을 제어하고 갱신된 값이 모듈을 통해 출력될 수 있도록 한다.
- FND 에 출력되는 값은 매 interval 마다 현재 0 이 아닌 값에서 1 증가된 값으로 갱신될 수 있도록 한다. 만약, 현재 digit 의 값이 8 이라면 다음 값은 9 가 아닌 1 이 되도록 설정하도록 한다. 초기에 입력 받았던 값을 기준으로 digit 이 증가하며 기존 값과 같은 값이 되는 경우 digit 의 위치를 한 칸 오른쪽으로 이동시킬 수 있도록 한다. 가령, 기존 값이 8 이었던경우 1 ~ 7 까지 1 바퀴를 돈 다음 8 로 갱신되는 순간, shift right 와 유사하게 FND 의 4 digit 을 기준으로 1 칸 오른쪽으로 이동될 수 있도록 한다. 현재의 0 이 아닌 digit 의 위치가 000X, 즉 가장 오른쪽이었던 경우 X000 이 될 수 있도록 한다.
- DOT 에 출력되는 값은 현재 FND 에 출력되는 00X0 과 같은 4 digit 중, 0 이 아닌 값과 같은 값을 DOT 모듈 상에 출력될 수 있도록 한다. 가령, 0020 이 현재 FND 에 출력된 상태라면 DOT 모듈에는 2 가 출력되어야 한다.
- LCD 는 총 32 length 이며 첫 16 character 가 LCD 의 첫 번째 줄, 다음 16 character 가 LCD 의 두 번째 줄에 출력되는 내용을 담게 된다. 첫 번째 줄에는 본인의 학번을 좌측 정렬, 응용 프로그램을 통해 전달받게 되는 TIMER_CNT 의 값을 우측 정렬로 표시될 수 있도록 한다. 가령, 본인의 학번이 “2018XXXX”, 전달 받은 TIMER_CNT 가 100 인 경우 “2018XXXX 100”이 될 수 있도록 하며, 이외의 칸은 ‘ ‘, 즉 공백으로 채워지도록 하여 가독성에 지장이 가지 않도록 한다. 이 중, TIMER_CNT 의 경우, ioctl 을 통해 구동 명령이 주어지는 경우, 매 interval 마다 값이 감소하게 되며 감소한 값에 맞게 적절히 right-align 하고 갱신된 값을 LCD 의 첫 번째 줄에 반영할 수 있도록 한다.
- LCD 의 두 번째 줄에는 본인의 이름 이니셜 3 자가 들어가게 된다. 이니셜의 경우 매 interval 마다 오른쪽/왼쪽으로 1 칸씩 이동될 수 있도록 한다. 초기에 이동하는 방향은 오른쪽으로 마지막 이니셜이 32 번째 index 에 오는 순간 이동 방향을 바꾸게 된다. 마찬가지로 이니셜이 왼쪽으로 1 칸씩 이동하는 경우, 첫 번째 이니셜이 16 번째 index 에 오는 경우 이동 방향을 바꾸게 된다.
- LED 의 경우, DOT 에 표현된 digit 에 해당되는 값과 같은 번호인 LED 가 turn on 될 수 있도록 한다.

- TIMER_CNT 회만큼 타이머 디바이스가 적절하게 수행되며 FPGA 모듈을 제어한 다음, FND 를 0000, DOT 와 LED 를 turn off 할 수 있도록 한다. LCD 의 경우 “Time’s up! 0₩nTurn off in x…”을 출력하도록 하며 이 중, x 는 3 으로 설정되어 3 초 후에 LCD 의 내용을 clear 하게 된다. X 의 경우 interval (= 1 second)마다 값이 감소하며 0 이 되는 순간 LCD 가 clear out 될 수 있도록 timer 를 설정한다.

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

2024.05.03: 명세서 정독

2024.05.04 ~ 2024.05.17: 타이머 디바이스 application, device driver 구현

2024.05.18: 보고서 작성

다. 개발 방법

A. 응용 어플리케이션

응용 어플리케이션에서는 target device 에서 host 에서 compile 한 실행파일을 실행할 때 TIMER_INTERVAL, TIMER_CNT, TIMER_INIT 의 값을 같이 argument 로 넘겨주게 된다.

이 때, argument 의 수가 총 4 개가 되어야 하고, TIMER_INTERVAL 은 1 ~ 100, TIMER_CNT 는 1 ~ 200, TIMER_INIT 은 0 이 3 개, 0 이 아니고 1 ~ 8 의 범위의 숫자 하나로 구성되어 있어야 한다. 해당 조건을 만족하지 못하는 경우, error message 가 출력될 수 있도록 하고 응용 프로그램을 종료한다.

```
if (argc < 4 || argc > 4) {
    printf("usage: ./app TIMER_INTERVAL[1-100] TIMER_CNT[1-200] TIMER_INIT[0001-8000]\n");
    return -1;
}
```

```
//check conditions. If conditions given to **argv is out of range, terminate program
if (TIMER_INTERVAL <= 0 || TIMER_INTERVAL > 100 || TIMER_CNT <= 0 || TIMER_CNT > 200) {
    printf("usage: ./app TIMER_INTERVAL[1-100] TIMER_CNT[1-200] TIMER_INIT[0001-8000, three zeros and one non-zero]\n");
    return -1;
}

int i, cnt = 0;
for(i = 0; i < 4; i++){
    if(argv[3][i] == '0') cnt += 1;
    if(argv[3][i] < '0' || argv[3][i] > '8'){
        printf("usage: ./app TIMER_INTERVAL[1-100] TIMER_CNT[1-200] TIMER_INIT[0001-8000, three zeros and one non-zero]\n");
        return -1;
    }
}

if(cnt != 3){
    printf("usage: ./app TIMER_INTERVAL[1-100] TIMER_CNT[1-200] TIMER_INIT[0001-8000, three zeros and one non-zero]\n");
    return -1;
}
```

입력에 문제가 없었을 경우, ioctl 을 통해 입력받은 TIMER_INTERVAL, TIMER_CNT, TIMER_INIT 을 바탕으로 타이머 디바이스의 초기화를 진행할 수 있도록 한다.

```
char options[15];
sprintf(options, "%03d %03d %04d", TIMER_INTERVAL, TIMER_CNT, TIMER_INIT);

ioctl(fd, IOCTL_SET, options);
```

이후, while()문과 read()를 통해 Achroimx 보드의 Reset 버튼이 눌러지는지 device driver 를 통해 모니터링을 하다가 Reset 입력이 주어지는 경우 while()문을 탈출하여 ioctl 을 통해 타이머 디바이스의 구동이 시작될 수 있도록 한다.

```
while(1){
    usleep(400000);
    int res = read(sw_dev, &push_sw_buff, sizeof(push_sw_buff));
    for(i=0;i<9;i++) {
        printf("[%d] ",push_sw_buff[i]);
    }
    printf(" res : %d\n", res);
    if(res == RESET_RECEIVED && push_sw_buff[0] == 80)
        break;
}
close(sw_dev);

fd = open(DEVICE_PATH, O_WRONLY);
ioctl(fd, IOCTL_START);
```

ioctl()을 성공적으로 전송했다면 close()를 통해 driver file 을 닫고 응용 프로그램이 종료될 수 있도록 한다.

B. 타이머 디바이스

타이머 디바이스의 작동을 제어할 디바이스 드라이버를 구현한다. 타이머 디바이스는 FND, DOT, LCD, PUSH_SWITCH, LED 를 제어할 수 있어야 한다. 이를 가능하게 하기 위해 디바이스 드라이버의 설치가 진행되는 init() 함수에서 FPGA 모듈들을 제어할 수 있도록 모듈의 physical address 와의 매핑을 진행한다.

```
/* physical mapping of fpga modules */
fpga_dot = ioremap(FPGA_DOT_ADDR, 0x10);
if(fpga_dot == NULL){
    printk(KERN_ERR "physical mapping unsuccessful.\n");
    return -1;
}

fpga_fnd = ioremap(FPGA_FND_ADDR, 0x02);
if(fpga_fnd == NULL){
    printk(KERN_ERR "physical mapping unsuccessful.\n");
    return -1;
}

fpga_lcd = ioremap(FPGA_LCD_ADDR, 0x01);
if(fpga_lcd == NULL){
    printk(KERN_ERR "physical mapping unsuccessful.\n");
    return -1;
}

fpga_led = ioremap(FPGA_LED_ADDR, 0x32);
if(fpga_led == NULL){
    printk(KERN_ERR "physical mapping unsuccessful.\n");
    return -1;
}

fpga_push_switch = ioremap(FPGA_PUSH_SWITCH, 0x18);
if(fpga_push_switch == NULL){
    printk(KERN_ERR "physical mapping unsuccessful.\n");
    return -1;
}
/* physical mapping end */
```

디바이스 드라이버의 등록을 진행하며 init()에서 register_chrdev()를 호출해 유효한 디바이스 드라이버로 등록되며 exit()에서의 unregister_chrdev()를 통해 유효한 디바이스 드라이버의 목록에서 삭제된다. 한편, 응용 프로그램에서 주어지는 명령에 대해 file_operations 를 정의하는 driver_fops 구조체에서 특정 작동에 실행되는 함수를 현재 구현하는 driver module 에 구현된 함수로 지정하여 해당 함수에 정의된 작동을 할 수 있게 된다.

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

타이머의 작동에 필요한 자료구조를 정의한다. 해당 과제에서는 timer 의 interval, 작동 횟수등이 주어지며 이를 interval, cnt 에 저장하도록 한다. 타이머가 interval 이 지남에 따라 FPGA 모듈의 값도 갱신되는데 이 때, FND 의 경우 현재 값이 00X0 이고 (X+1) % 8 ~ (X-1) % 8 의 범위로 갱신되었다면 다음 갱신에는 000X 로 갱신되어야 한다. 이를 수월하게 관리할 수 있도록 init 을 선언했고 해당 변수에서 TIMER_INIT 으로 전달되는 값을 관리하게 된다.

그 외에도 LCD 의 제어에 필요한 본인의 학번, 이니셜을 관리하기 위해 st_id 와 name_initial 을 선언했고 first_initial, last_initial, dir 을 통해 현재 LCD 가 타이머의 interval 이 지남에 따라 어느 방향으로 갱신될 지 관리할 수 있도록 했다.

```
//Timer data structure
typedef struct _timer{
    struct timer_list timer;
    int interval;
    int cnt;
    int cnt_tmp;
    char init[5];
    int origin_num;

    char st_id[9];
    char name_initial[4];
    char lcd_timer_cnt[4];
    char first_initial;
    char last_initial;
    int start_idx;
    int dir;
    int lcd_timer_cnt_len;
}TIMER;

TIMER timer, timer2;
```

driver_ioctl()함수에서는 응용 프로그램에서 ioctl() 명령이 주어지는 경우, 전달된 parameter 중 cmd 의 값에 맞게 작동이 처리될 수 있도록 한다. 명령에는 두 가지 종류가 있고 각각 IOCTL_SET 과 IOCTL_START 이다.

```
//MACRO for ioctl command
#define IOCTL_SET _IOW(DEVICE_MAJOR_NUMBER, 0, char*)
#define IOCTL_START _IO(DEVICE_MAJOR_NUMBER, 0)
```

IOCTL_SET 은 응용 프로그램을 통해 전달된 TIMER_INTERVAL, TIMER_CNT, TIMER_INIT 을 값을 이용해 앞서 언급한 타이머에서 사용하는 자료구조의 초기화를 진행한다. 응용 프로그램에서 전달되는 string 에 해당 정보들이 저장되어 있고, 적절히 3, 3, 4 의 길이를 토대로 자료구조에 저장될 수 있도록 한다.

```
/* Parse TIMER_CNT passed from user application */
for(i = 6; i >= 4; i--){
    lcd_str[j] = str[i];
    timer_cnt[j] = str[i];
    j--;
}

/* Parse TIMER_INTERVAL passed from user application */
j = 0;
char timer_interval_str[4];
char timer_init[5];
for(i = 2; i >= 0; i--){
    timer_interval_str[i] = str[i];
}
```

```
/* Parse TIMER_INIT passed from user application */
j = 0, timer_val = 0;
for(i = 8; i < 12; i++){
    timer_init[j] = str[i];
    if(timer_init[j] != '0') timer_val = timer_init[j] - '0';
    printk("%c", timer_init[j]);
    j++;
}
timer_init[4] = '\0';
```

TIMER_CNT 로 전달되는 값을 통해 해당 문자열의 길이를 구한다. 해당 정보는 후에 interval 이 지나며 LCD 에 표시되는 TIMER_CNT 의 정보를 갱신하는 것에 사용된다. 학번, 이니셜 등을 LCD 의 갱신에서 사용될 수 있도록 자료구조의 초기화를 진행하며 해당 작업을 마쳤다면 타이머 디바이스에서 사용할 fpga 모듈들의 초기화와 타이머의 초기화 진행될 수 있도록 한다.

```
//timer_cnt_len is used to update TIMER_CNT shown on LCD.
printk("timer count : %d\n", timer_count);
int timer_cnt_len;
if(timer_count < 10){
    timer_cnt_len = 0;
}
else if(timer_count < 100){
    timer_cnt_len = 1;
}
else
    timer_cnt_len = 2;
```

```
// /* Initialize timer values */
printk("Setting initial values of timer\n");
timer.interval = timer_interval;
timer.cnt = timer_count;
strcpy(timer.init, timer_init);
timer.origin_num = timer_val;
strcpy(timer.st_id, "20181625");
strcpy(timer.name_initial, "NHJ");
strcpy(timer.lcd_timer_cnt, timer_cnt);
timer.dir = 2;
timer.first_initial = timer.name_initial[0];
timer.last_initial = timer.name_initial[2];
timer.start_idx = 16;
timer.lcd_timer_cnt_len = timer_cnt_len;
printk("Setting initial values of timer done\n");
/*  Timer initialization done */
```

IOCTL_START 의 경우, 응용 프로그램에서 명령이 주어질 때 실제로 구동이 진행되도록 하며 driver_set()을 호출하는 것으로 timer 를 등록해 interval 이 지날 때마다 timer 에 등록된 function 의 작동이 수행될 수 있도록 한다.

```
case IOCTL_START:
    /* start timer device */
    driver_set();
    break;
```

```c
/*
    driver_set(void) : Initialize timer and add timer to timer list.
*/
void driver_set(){
    printk("driver_set()\n");
    /*
        Initialize timer and add timer to timer list
    */
    init_timer(&timer.timer);
    timer.timer.data = (unsigned long)&timer;
    timer.timer.expires = get_jiffies_64() + timer.interval * (HZ / 10);
    timer.timer.function = timer_blink;
    add_timer(&timer.timer);
}
```

Timer 에 등록되는 function 인 timer_blink()에서는 매 interval 마다 FPGA 모듈의 값을 갱신할 수 있도록 한다. 각각 fpga_fnd_blink(), fpga_lcd_blink(), fpga_dot_blink(), fpga_led_blink()로 모듈의 제어를 실행하며 이는 timer_cnt 의 값이 1 이상일 때까지 진행된다.

```c
//If timer interrupt handling is executed less than TIMER_COUNT times, add timer to timer list.
if(timer_data->cnt >= 1){
    timer_data->timer.data = (unsigned long)&timer;
    timer_data->timer.expires = get_jiffies_64() + timer_data->interval * (HZ / 10);
    timer.timer.function = timer_blink;

    add_timer(&timer_data->timer);
}
```

다만, timer_cnt 가 0 이 되는 경우, 질의응답란에 답변을 참조하여 해당 timer 의 interval 이 지난 다음 “Time’s up!…”와 같은 종료 안내 메시지가 출력되는 것이 아닌, timer_cnt 가 만료된 직후 종료 안내 메시지가 출력될 수 있도록 time interval 이 아닌 0.2 초의 expiration 을 주었다.

```
else{   //timer expired. Set FND to 0000, clear DOT, turn of LED. Set LCD to "Time's up!....." string and set up another timer to handling closing message.
    //fpga_lcd_blink();
    char fnd_str[5] = {0,};
    // char lcd_str[33] = {' ',};
    // strcpy(lcd_str, "Time's up!     ");
    // lcd_str[15] = '0';
    timer2.cnt = 3;
    // sprintf(lcd_str+16, "Shutdown in %d...", timer2.cnt);
    // fpga_dot_init(0);
    fpga_fnd_init(fnd_str);
    // fpga_lcd_init(lcd_str);
    fpga_dot_clear();
    fpga_led_init(0);
    init_timer(&timer2.timer);
    timer2.timer.data = (unsigned long)&timer2;
    timer2.timer.expires = get_jiffies_64() + 2 * (HZ / 10);
    timer2.timer.function = timer_after_timer_cnt;

    // timer.timer.data = (unsigned long)&timer;
    // timer.timer.expires = get_jiffies_64() + timer.interval * (HZ / 10);
    // timer.timer.function = timer_blink;
    add_timer(&timer2.timer);
    del_timer(&timer);
    //add_timer(&timer_data->timer);
    // fpga_led_init(0);
    printk("timer_blink() expired\n");
}
```

이후의 timer 는 새로운 timer 인 timer2 로 해당 timer 를 timer list 에 등록 후, function 으로 timer_after_timer_cnt()를 선언하여 실행할 수 있도록 했다.

```
static void timer_after_timer_cnt(unsigned long data){
    printk("timer_after_timer_cnt() called\n");
    TIMER* timer_data = (TIMER*)data;
    timer_data->cnt--;
    printk("kernel_timer_blink %d\n", timer_data->cnt);
    fpga_lcd_blink_after_timer();

    //if timer interrupt handling executed less than timer cnt times, add timer to timer list.
    if(timer_data->cnt >= 0){
        timer_data->timer.data = (unsigned long)&timer2;
        timer_data->timer.expires = get_jiffies_64() + 10 * (HZ / 10);
        timer.timer.function = timer_after_timer_cnt;

        add_timer(&timer_data->timer);
    }
    else{
        char fnd_str[5] = {0, };
        char lcd_str[33] = {0, };
        // fpga_dot_init(0);

        /* blank out FND & LCD */
        fpga_fnd_init(fnd_str);
        fpga_lcd_init("                                ");
        // fpga_led_init(0);


        //delete timer
        del_timer(&timer2);
        printk("Timer expired\n");
    }
}
```

timer_after_timer_cnt()에서는 LCD 에서 종료 되기 x 초 전이라는 메시지가 갱신될

수 있도록 했고 x 는 3 부터 시작하여 interval 인 1 초가 지날 때마다 1 씩 감소하게 되고 timer_cnt 가 0 이 된다면 LCD 를 blank line 으로 초기화하며 타이머 디바이스의 작동이 종료된다.

```
static void timer_after_timer_cnt(unsigned long data){
    printk("timer_after_timer_cnt() called\n");
    TIMER* timer_data = (TIMER*)data;
    timer_data->cnt--;
    printk("kernel_timer_blink %d\n", timer_data->cnt);
    fpga_lcd_blink_after_timer();

    //if timer interrupt handling executed less than timer cnt times, add timer to timer list.
    if(timer_data->cnt >= 0){
        timer_data->timer.data = (unsigned long)&timer2;
        timer_data->timer.expires = get_jiffies_64() + 10 * (HZ / 10);
        timer.timer.function = timer_after_timer_cnt;

        add_timer(&timer_data->timer);
    }
    else{
        char fnd_str[5] = {0, };
        char lcd_str[33] = {0, };
        // fpga_dot_init(0);

        /* blank out FND & LCD */
        fpga_fnd_init(fnd_str);
        fpga_lcd_init("                                ");
        // fpga_led_init(0);


        //delete timer
        del_timer(&timer2);
        printk("Timer expired\n");
    }
}
```

Ⅳ. 연구 결과

- 최종 연구 개발 결과를 자유롭게 기술할 것.

강의자료 4 의 page 28 에 있는 overall flow of kernel timer handling 슬라이드의 내용을 바탕으로 timer interrupt 발생 시의 처리 과정을 구현하고 수행 결과를 확인할 수 있었다.

Timer 는 timer 가 사용하는 자료구조에서 count 를 비교하여 새로운 timer 를 등록하거나, timer 를 종료시킨다. 이 과정에서, get_jiffies_64() + timer interval 를 통해 현재부터 어느 정도가 지나야 timer 를 expire 할지 설정하고 한 timer 에서 다른 timer 의 등록도 가능하다는 것을 확인할 수 있었다.

```
else{
    //fpga_lcd_blink();
    char fnd_str[5] = {0,};
    // char lcd_str[33] = {' ',};
    // strcpy(lcd_str, "Time's up!     ");
    // lcd_str[15] = '0';
    timer2.cnt = 3;
    // sprintf(lcd_str+16, "Shutdown in %d...", timer2.cnt);
    // fpga_dot_init(0);
    fpga_fnd_init(fnd_str);
    // fpga_lcd_init(lcd_str);
    fpga_dot_clear();
    fpga_led_init(0);
    init_timer(&timer2.timer);
    timer2.timer.data = (unsigned long)&timer2;
    timer2.timer.expires = get_jiffies_64() + 2 * (HZ / 10);
    timer2.timer.function = timer_after_timer_cnt;

    // timer.timer.data = (unsigned long)&timer;
    // timer.timer.expires = get_jiffies_64() + timer.interval * (HZ / 10);
    // timer.timer.function = timer_blink;
    add_timer(&timer2.timer);
    del_timer(&timer);
    //add_timer(&timer_data->timer);
    // fpga_led_init(0);
    printk("timer_blink() expired₩n");
}
```

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.

프로젝트를 수행하며 디바이스 드라이버의 활용을 통한 하드웨어 제어를 조금 더 자세하게 이해할 수 있었다.

하지만, 요구사항 명세서에서 조금씩 아쉬운 부분이 있었는데 조금 더 명확하게 타이머 디바이스가 TIMER_CNT 만큼 작동된 다음, 이후의 종료 안내 메시지의 처리가 진행되는지 적혀 있었다면 좋았을 것 같다.