Embedded System Software HW #2

due date: 2019. 5. 16.

student name: 윤제형
student id: 20151575

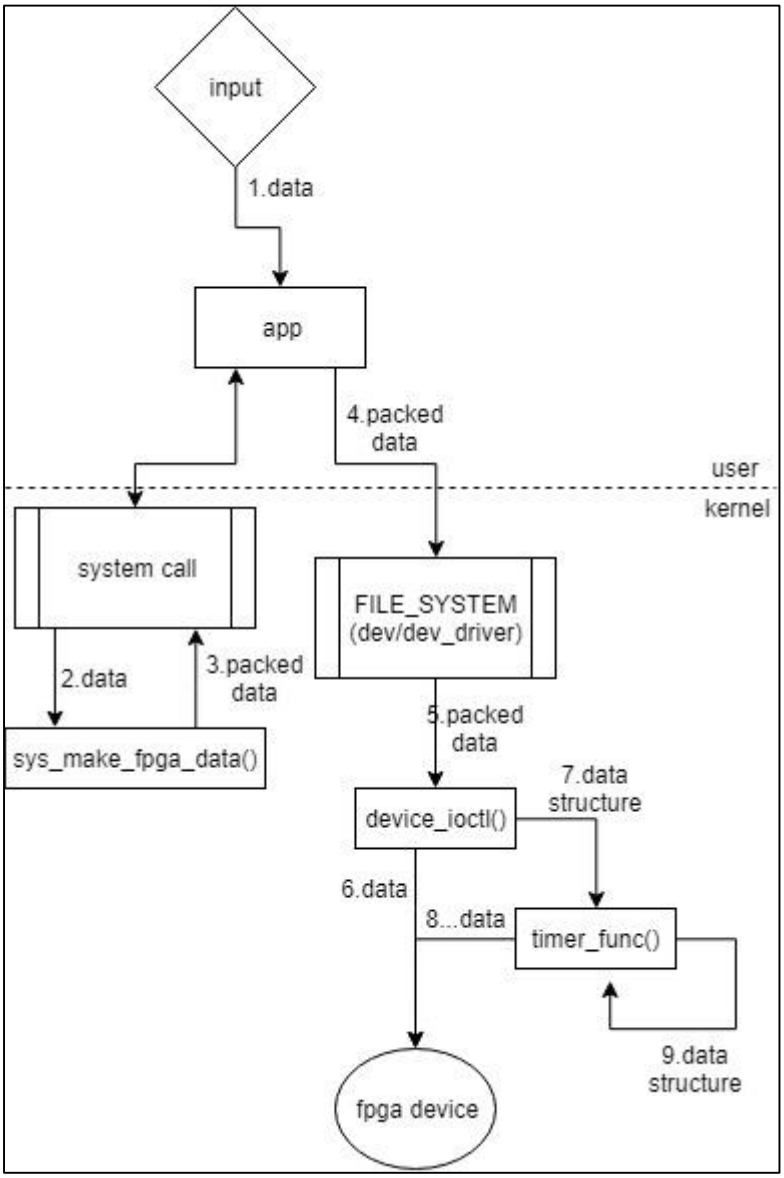
목차

1. 목표

- 수업시간에 배운 system call 과 module, device driver 을 기억하여 fpga device들을 control 할 수 있는 logic을 설계하고 구현한다

- 구현 대상은 다음과 같다. 1). fpga device들을 control할 module을 작성한다. 2). 1)에서 작성한 module에 넘겨줄 data를 packing 하여 4byte data 로 만드는 system call을 구현한다. 3). 2)에서 구현한 system call 을 활용하여 data를 제작하고 이를 1)에서 구현한 module을 통해 fpga device를 제어 할 수 있는 user 응용 프로그램 제작 이다.

2. 순서도



- 제작된 프로그램의 순서도는 위와 같다. User level program 인 app 이 system call 을 통해 interrupt 를 호출하여 미리 커널에 등록 시켜놓은 sys_make_fpga_data() 입력 받은 data 를 그대로 넘겨준다. 그러면 kernel 은 sys_make_fpga_data() 를 호출하여 수행하고 넘겨 받은 data 를 규칙에 맞게 packing 한다. Packing 이 끝난 뒤 이를 다시 user 로 반환하고 user 는 넘겨받은 이 packed data 를 device 에 쓰기 위해 device file 을 열고 device file 에 packed data 를 알맞은 handling function 을 통해 넘겨준다. 그러면 kernel 이 data 를 device_ioctl()에 전달하고 이를 실행 시켜준다. Device_ioctl()은 넘겨 받은 packed data 를 다시 개별적인 data 들로 분리하고 이를 data structure 에 저장한다 이후 data 에 맞게 fpga device 들에 write 를 진행하고 timer logic 을 통해 timer_func() 을 time interval 에 맞게 호출 할 수 있고 매개변수로 data_structure 가 넘어 갈 수 있게 지정한다. 이후 timer_func() 은 시간 마다 호출되어 적절한 device_ioctl() 에서 했던 것 과 비슷하게 적절한 출력을 진행한다.

3. 추가 구현과 구현 방법

A. Systemcall, 4byte data

```
3   asmlinkage unsigned int sys_make_fpga_data(int interval,int count, int val) {
```

System call은 이와 같이 timer 의 interval 총 반복 횟수, 시작 fnd값을 입력 받는다.
이후 fnd data에서 처음으로 value가 나온 index와 그 value를 추출하여 새로 정한 포맷에 맞게 아래와 같이 packing 한 후 이를 반환한다.

```
t_idx =  (t_idx << (8*3)) & 0xFF000000;
t_val =  (t_val << (8*2)) & 0x00FF0000;
t_cnt =  (t_cnt << (8*1)) & 0x0000FF00;
t_intval =  t_intval & 0x000000FF;

data =  t_idx | t_val | t_cnt | t_intval;
    return data;
```

새로 정의한 data의 packing 정보는 다음과 같다.

Fnd 인덱스 정보	Fnd 값 정보	반복 횟수 정보	Interval 정보

각 칸은 1byte의 크기를 가지고 있다.

B. Ioctl()

```
long device_ioctl(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param){
```

Ioctl 함수는 위에서 작성한 system call 을 통해 구성한 data를 매개변수 ioctl_param을 통해 넘겨 받는다.

```
get_user(data,(unsigned int *)ioctl_param);
start_idx = (data >> (8*3)) & 0xFF;
start_val = (data >> (8*2)) & 0xFF;
count = (data >> (8*1)) & 0xFF;
interval = (data >> (8*0)) & 0xFF;
```

이를 위와 같이 파싱하여 원하는 데이터를 저장한다. 이 파싱 한 정보는 일부는 device에 write되고 다른 일부는 새롭게 정의한 data structure인

```
typedef struct _device_timer{
    struct timer_list timer;
    unsigned int count,interval,start_val,idx;
    short first_text_idx,second_text_idx;
    short first_text_dir,second_text_dir;
}_device_timer;
```

에 알맞게 저장하여 timer logic에 따라 호출될 함수의 매개변수로 passing 한다..
Timer logic은 Timer에 설정된 time interval에 따라

```
static void timer_func(unsigned long timeout){
```

를 호출하게 되고 이는 알맞게 데이터를 출력하고 timer logic을 통해 interval 에 따라 또 timer func이 호출되어 알맞게 데이터를 출력한다. 그리고 따로 ioctl의 경우가 나누어 지지 않으므로 ioctl_num을 통한 경우의 수를 나누지는 않았다.

C. APP

```
    unsigned int data = syscall(376,interval,count,val);
    dev = open("/dev/dev_driver",O_WRONLY);
    if(dev <0 ){
        printf("open error!\n");
        return -1;
    }
    ioctl(dev,_IOR(242,0,unsigned int),&data);
    close(dev);
    return 0;
}
```

위에서 언급한 system call을 통해 packed data를 넘겨 받아 저장하여 device를 open 하고 ioctl의 data로 이를 passing 하였다. loctl num으로서 major number를 명시한 num을 생성하여 이를 passing 하였다.