Embedded System Software 과제 2

(과제 수행 결과 보고서)

과목명:

담당교수:

학번 및 이름:

개발기간:

최 종 보 고 서

I. 개발 목표

Embedded linux 커널상에서 동작하는 타이머 디바이스 드라이버 모듈을 개발한다. 또한 이를 테스트할 수 있는 테스트 응용 프로그램을 개발한다.

II. 개발 범위 및 내용

가. 개발 범위

타이머 디바이스 드라이버 모듈을 개발한다. 이때 타이머 기능을 위해 kernel timer를 사용한다. 또한 fpga의 fnd, led, dot matrix, text lcd를 사용한다. 이를 사용하는 데에 있어서 memory mapped IO 방식을 이용한다.

테스트 응용 프로그램을 개발한다. 이때 타이머 디바이스 드라이버를 사용하기 위해 디바이스 파일 입출력을 이용한다. 특히 사용자 입력으로 받은 타이머 옵션을 전달하기 위해 ioctl을 사용하고 타이머를 구동하기 위해 ioctl을 사용한다.

나. 개발 내용

테스트 응용 프로그램

사용자로부터 timer interval, timer cnt, timer init을 입력으로 받는다. 각 입력의 의미는 다음과 같다.

- TIMER_INTERVAL HZ 값 1~100 (0.1~10 초)
- TIMER_ CNT : 디바이스 출력 변경 횟수 (1~100)
- TIMER_ INIT : FND 에 출력되는 초기 문양과 위치 0001~8000

이 입력을 ioctl을 사용하여 타이머에 전달하고 ioctl을 사용하여 타이머를 구동한다.

타이머 디바이스 드라이버

사용자 입력으로 받은 옵션을 기준으로 4가지 디바이스(fnd, led, dot matrix, text lcd)에 적절히 출력한다. 모든 디바이스의 출력은 timer interval을 기준으로 바뀐다. 모든 디바이스는 timer cnt 횟수만큼 출력한다.

fnd

초기 출력: timer init 그대로 출력한다.

마지막 출력: 모든 출력이 끝난 이후 0000을 출력한다.

다음 문양이란 현재 문양에서 1 증가한 수를 의미한다. 단 8의 다음 문양은 1이다.

문양이 출력되는 위치는 한 번의 로테이션이 끝날 때마다 우측으로 이동한다. 단 4번째 자리에서 로테이션이 끝났다면, 1번째 자리로 이동한다.

led

현재 fpga_fnd에서 출력 중인 문양의 번호를 나타낸다.

TIMER_CNT 횟수만큼의 출력이 끝나면 fpga_led의 불을 꺼준다.

dot

현재 fpga_fnd에서 출력 중인 문양과 같은 모양의 문양을 출력한다.
TIMER_CNT 횟수만큼의 출력이 끝나면 dot의 불을 꺼준다.

text lcd

초기 상태: 첫 번째 줄에는 자신의 학번을 출력하고, 두 번째 줄에는 자신의 이름을 영문으로 출력한다. 이때 두 문장 모두 left align 시킨다.
TIMER_INTERVAL마다 오른쪽으로 한 칸씩 shift 하고, 가장 오른쪽 칸에 도달한 경우, 다시 왼쪽으로 shift 이동을 시작한다. 가장 왼쪽 칸에 도달한 경우, 다시 오른쪽으로 shift 이동을 시작한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

05.03-05.09 : module 함수 개발
05.10-05.15 : submodule 함수 개발
05.16 : 보고서 작성

나. 개발 방법

테스트 응용 프로그램

app폴더 내의 app.c파일을 개발한다. 타이머 디바이스 드라이버를 open, ioctl, close한다. 두가지 ioctl을 구분하기 위하여 cmd를 이용한다.

타이머 디바이스 드라이버

module폴더 내의 dev_driver.c와 dev_driver.h를 개발한다. dev_driver.c에는 module을 위한 init, exit 함수와 디바이스 드라이버를 위한 open, release, ioctl함수가 있다. 타이머를 사용하기 위해 kernel timer를 사용한다. 이를 위한 handler함수 또한 개발한다.
dev_driver.h에는 app.c에서도 사용하는 struct dev_driver_data가 정의되어 있고 타이머 디바이스 드라이버의 이름과 major number, 그리고 ioctl의 두가지 cmd가 정의되어 있다.
4가지 디바이스들을 사용하기 위하여 submodule폴더내의 submodule.c, submodule.h를 개발한다. 디바이스들은 memory mapped io를 이용하여 컨트롤한다. 이는 ioremap, iounmap 함수를 이용한다.

IV. 연구 결과

주요 파일들의 주요 함수들은 다음과 같다.

dev_driver.c

int dev_driver_open(struct inode* minode, struct file* mfile) : 4가지 디바이스들을 컨트롤 하기 위하여 각 디바이스 물리주소를 kernel 가상주소에 매핑한다. 즉 submodule_ioremap을 호출한다.
int dev_driver_release(struct inode* minode, struct file* mfile) : close가 호출될때 호출된다. 특별한 동작은 하지 않는다.
long dev_driver_ioctl(struct file* mfile, unsigned int cmd, unsigned long _data) : 두가지 cmd에 따라 동작이 다르다. CMD0일 경우, argument로 받은 데이터를 사용하여 4가지 디

바이스들을 초기화한다. 즉 submodule_init을 호출한다. 그리고 kernel timer를 초기화하기 위하여 init_timer를 호출한다. CMD1일 경우, kernel timer를 구동하기 위하여 add_timer를 호출한다.
int __init dev_driver_init(void) : 드라이버 함수들을 등록하기 위하여 register_chrdev를 호출한다.
void __exit dev_driver_exit(void) : unregister_chrdev를 호출한다. 또한 할당된 가상주소를 해제하기 위하여 submodule_iounmap을 호출한다.
static void dev_driver_handler(unsigned long _mydata) : kernel timer의 handler 함수이다. 4가지 디바이스들의 다음 상태를 출력한다. 즉 submodule_update를 호출한다. 그 후 add_timer하여 다시 타이머를 구동한다. 이를 cnt에 맞게 반복한 후 모든 출력이 끝났을때, 디바이스들이 마지막 상태를 출력하도록 한다. 즉 submodule_clear를 호출한다.

submodule.c

4가지 디바이스들을 컨트롤하기 위한 함수들이 있다. 대표적으로 다음 함수들이 있다.
int submodule_ioremap(void) : 디바이스들을 ioremap을 사용하여 kernel 가상메모리상에서 접근할 수 있도록 한다.
int submodule_iounmap(void) : iounmap을 사용하여 할당된 가상메모리 주소를 해제한다.
int submodule_init(char*) : 디바이스들이 초기상태를 출력하도록 한다.
int submodule_update(void) : 디바이스들이 다음 상태를 출력하도록 한다. 이때 이를 위해 필요한 변수들은 static global 변수로 선언되어 있다. 따라서 이 함수는 따로 argument를 받을 필요가 없다.
int submodule_clear(void) : 디바이스들이 마지막 상태를 출력하도록 한다.

V. 기타

다양한 디바이스(fnd, led, dot, text lcd)를 ioremap을 사용하여 컨트롤하는 경험을 할 수 있어서 좋았다. kernel timer를 사용해보는 것도 재미있었다. static global 변수를 사용하면 해당 파일안에 존재하는 함수를 호출할때 따로 argument를 관리해주지 않아도 된다는 점을 이용하여 편리하게 코딩했다. c에서 객체지향의 private 변수를 사용하는 것과 비슷하다는 느낌을 받았다. 다만 global 변수이기 때문에 동적으로 해제가 안된다는 점에서 차이가 있다. 디버깅을 할때 어느 함수에서 어느 조건일때 버그가 발생하는지를 먼저 찾는 것이 중요하다는 생각이 들었다.