**Embedded 2nd HW.**

**(설계 프로젝트 수행 결과)**

**과목 명: [CSE4116] Embedded Systems**

**담당 교수: 서강대학교 컴퓨터 공학과 박 성 용**

**학번 및 이름: 20091648, 이 준 호**

**개발 기간: 2014. 05. 07. - 2014. 05. 24.**

**최종 보고서**

**I. 개발 목표**

본 프로젝트에서는 system call programming, module programming, device driver 구현 등을 활용하여 achro 4210 Exynos 보드 위에서 작동하는 module, system call과 이를 이용한 application을 구현한다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

Device Driver - 기존의 device driver (gpio\_fnd, gpio\_led, fpga\_fnd, fpga\_led, fpga\_dot, fpga\_text\_lcd) 와 timer module을 포함하는 하나의 module을 구현한다. 각 device는 최초 한번 initialize가 되고, user level에서 write 함수를 한번만 호출하면 각 device에 맞는 데이터가 타이머를 기준으로 동시에 출력 되어야 한다.

System call - Parameter를 받아서 하나의 변수로 return 해주는 system call을 구현한다. 사용자는 시간 간격, 횟수, 시작 옵션의 3가지 parameter를 argument로 system call에게 넘겨주고 system call은 적절한 encoding 과정을 거쳐서 4 byte stream으로 그 값을 return 해준다.

Application - 위에서 구현한 device driver와 system call을 모두 이용하여 간단한 출력을 해주는 application을 구현한다. 이 어플리케이션은 device 파일을 open 하여, write를 통해 출력을 시작해 준 뒤, close하고 종료된다.

**나. 개발 내용**

모든 device의 값들이 바뀌는 시간은 타이머를 기준으로 한다.

1) Device Driver 및 Timer

- gpio\_fnd

FND 4자리를 출력으로 컨트롤하고, 위와 같은 순서의 문양을 timer에 따라 반복해서 출력하게 한다. 출력하는 위치는 한 번의 로테이션이 끝날 때마다 우측으로 이동한다. (ex. 3번째 자리에서 1번 문양부터 시작하였을 때, 8번 문양까지 출력 한 뒤, 4번째 자리로 이동하여 다시 1번 문양부터 8번 문양까지 출력한다) 지정 횟수 만큼의 출력이 끝나면 FND의 불을 꺼준다.

- gpio\_led

gpio\_fnd가 현재 문양을 출력하는 위치의 바로 아래 LED를 점등한 상태로 꺼지지 않고, FND의 문양 출력 위치가 바뀔 경우 LED의 위치도 함께 바뀐다. (ex. 1번 FND가 문양을 출력 중이라면, 바로 아래 있는 1번 LED가 점등 되고, 1번 FND의 출력이 끝나고, 2번 FND에서 출력이 시작될 시에 LED도 1번 LED가 꺼지고, 2번 LED가 점등 된다) 모든 문양의 출력이 끝나면 LED의 불을 꺼준다.

- fpga\_led

현재 gpio\_fnd에서 출력 중인 문양의 번호를 나타낸다. (D1 : 1번, D2 : 2번, D3 : 3번, D4 : 4번, D5 : 5번, D6 : 6번, D7 : 7번, D8 : 8번) 모든 문양의 출력이 끝나면 fpga\_led의 불을 꺼준다.

- fpga\_fnd

초기 값은 문양이 출력 되는 횟수이고, 한 문양이 출력 될 때마다 1씩 감소한다. 모든 문양의 출력이 끝나면 0으로 초기화 된다.

- fpga\_dot

현재 gpio\_fnd에서 출력 중인 문양과 같은 모양의 문양을 출력한다. gpio\_fnd의 문양이 바뀐다면, fpga\_dot도 gpio\_fnd와 같은 문양으로 함께 바뀐다. 지정 횟수 만큼의 출력이 끝나면 dot의 불을 꺼준다.

- fpga\_text\_lcd

첫 번째 줄에는 자신의 학번을 입력하고, 두 번째 줄에는 자신의 이름을 영문으로 입력한다. 두 줄 모두 timer에 따라 오른쪽으로 한 칸씩 shift 이동을 하고, 오른쪽 칸에 더 이상의 공백이 없을 경우, 왼쪽으로 shift 이동을 시작한다. 왼쪽 shift 이동도 마찬가지로, 왼쪽 칸에 더 이상의 공백이 없을 경우, 오른쪽으로 shift 이동을 시작한다. 이 과정은 모든 문양의 출력이 끝날 때까지 계속되고, 종료 시 text\_lcd를 초기화 시킨다.

- Timer

Timer를 기준으로 모든 device의 출력 시간을 결정한다.

2) System call

입력 값을 받아서, write() 함수에 넣어줄 데이터를 return 한다.

입력 값 : 시간 간격, 횟수, 시작 옵션

리턴 값 : 4 byte stream (1byte: 시작 fnd 위치, 1byte: 시작 fnd 값, 1byte: 시간 간격, 1byte: 횟수)

|  |  |  |  |
| --- | --- | --- | --- |
| 시작 fnd 위치 | 시작 fnd 값 | 시간 간격 | 횟수 |

1 byte 1 byte 1 byte 1 byte

3) 응용 프로그램 (application)

device 파일을 open하여, write를 통해 출력을 시작해준 뒤, close하고 종료된다.

실행 예) ./app 시간 간격[1-100] 횟수[1-100] 시작 옵션[0001-8000]

시간 간격 : 0.1 - 10초

횟수 : 타이머 호출 횟수 (8->한번의 로테이션)

시작 옵션의 의미 : 0040 -> 3번째 자리에서 4번째 문양부터 출력을 시작한다.

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

|  |  |
| --- | --- |
| 2014. 05. 07. | Assignment #2에서 요구하는 요구 사항 분석 및 프로젝트 계획 구성 |
| 2014. 05. 08. | 주어진 device driver 와 application example 분석 |
| 2014. 05. 13. | Module programming 스터디 |
| 2014. 05. 15. | System call 및 application 스터디 |
| 2014. 05. 20. | Module 및 응용 프로그램의 기본적인 틀 구성 |
|  | System call 개발 |
| 2014. 05. 21. | Kernel에 system call 추가 및 응용 프로그램에서의 작동 여부 확인 |
|  | 주어진 device driver 을 하나의 module로 통합 |
| 2014. 05. 22. | 개발한 device module을 board 상에서 test 작업 수행 |
|  | 응용 프로그램 및 system call 정상 작동 여부 확인 |
|  | 모든 요구 사항 충족 여부 확인 |
| 2014. 05. 23. | Code 최적화 및 정리 |
|  | Document 1차 초안 작성 |
| 2014. 05. 24. | Document 수정 및 검토 |
|  | 프로젝트 파일 및 document 제출 |

**나. 개발 방법**

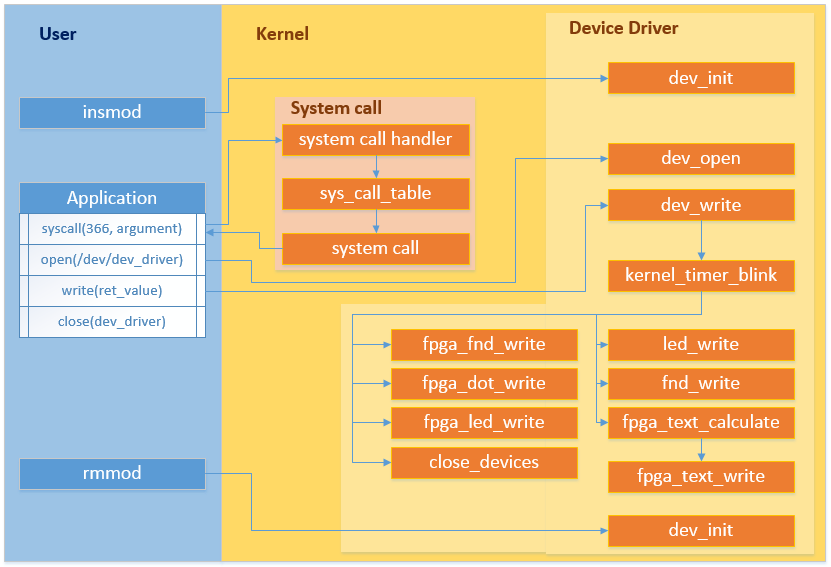
Module programming : 여러 device driver를 동일한 major number로 initialize 한 후 write 함수로 넘겨 받은 4 byte stream 입력 값을 decode 하여 각 device에 맞는 값들을 넘겨 준다. 각 device 들은 넘겨 받은 데이터를 board 상에서 출력을 해준다. 이 때, 각 device 들은 timer에 의해 동시에 바뀌어야 하기 때문에 timer가 반복 적으로 호출되는 구간에서 각 device에 값을 넘겨주도록 구현해야 한다. 또한 지정된 시간 만큼의 시간이 경과 되면 각 device는 초기 상태로 돌아가 board 상에서 아무것도 출력하지 않아야 한다.

Implementing system call : kernel 내의 unistd.h 파일에 system call number를 할당하고, call.S 파일에 system call을 system call table에 등록 시킨다. kernel 폴더에 새로운 system call 함수를 만들고, kernel을 컴파일 한 후 board 상에 올려서 작동 여부를 확인한다.

Application : 3개의 argument를 받아 system call을 이용해 4 byte stream을 리턴 받고 그 값을 다시 device driver에 write 하여준다. 모든 작업이 끝나면 device driver를 close하고 프로그램을 종료한다.

**IV. 연구 결과**

**1. 합성 내용:**



**2. 제작 내용: 개발 결과**

**int dev\_open(struct inode \*, struct file \*)**

역할 : Device driver의 usage counter를 설정한다.

구현 방법 : 전역 변수 dev\_uasge를 1로 설정한다.

**int dev\_release(struct inode \*, struct file \*)**

역할 : Device driver의 usage counter 설정을 해제한다.

구현 방법 : 전역 변수 dev\_usage를 0으로 설정한다.

**int close\_devices(void)**

역할 : fnd driver, led driver, fpga led driver, fpga fnd driver, fpga dot driver, fpga text driver를 deafult 값으로 설정한다.

구현 방법 : 각 driver의 argument 값으로 0을 passing 한다.

**static void kernel\_timer\_blink(unsigned long)**

역할 : timer가 사용자가 지정한 time interval 간격으로 종료 조건 시간까지 반복된다.

구현 방법 : count가 end\_count와 보다 커질 경우 함수가 종료되며, timer에 스케쥴링 시킨다.

**unsigned short fnd\_write(const unsigned short \*)**

역할 : gpio fnd에 해당하는 값을 출력한다.

구현 방법 : parameter로 넘어온 argument를 decode 하여 그에 맞는 위치와 값을 fnd device에 outb()를 사용하여 출력한다.

**ssize\_t led\_write(const char \*)**

역할 : gpio led에 해당하는 값을 출력한다.

구현 방법 : parameter로 넘어온 argument에 맞는 위치에 LED에 outb()를 사용하여 불을 켜준다.

**ssize\_t fpga\_led\_write(const char \*)**

역할 : fpga led에 현재 출력 되고 있는 character의 순서에 맞는 LED를 켜준다.

구현 방법 : parameter로 넘어온 argument에 해당하는 LED에 outb()를 사용하여 불을 켜준다.

**ssize\_t fpga\_fnd\_write(const int \*)**

역할 : fpga fnd에 남은 횟수를 출력하여 준다.

구현 방법 : parameter로 넘어온 argument의 값을 string 형으로 변환 시켜 outb()를 사용하여 fpga fnd에 값을 출력하여 준다.

**ssize\_t fpga\_dot\_write(const char \*)**

역할 : fpga dot에 현재 출력 중인 character를 출력하여 준다.

구현 방법 : parameter로 넘어온 argument의 index를 outb()를 사용하여 dot driver에 출력하여 준다.

**int fpga\_text\_calculate(int)**

역할 : fpga text lcd에 이름과 학번이 좌우로 shift 되며 출력하도록 계산한다.

구현 방법 : direction flag를 사용하여 좌우 방향을 구분하고 각 text lcd의 끝 부분에 도달하였을 경우 direction을 반대로 변경하여 반대로 shift 될 수 있도록 한다.

**ssize\_t fpga\_text\_write(const char \*)**

역할 : fpga\_text\_calculate에서 넘어온 string을 fpga text lcd에 출력하여 준다.

구현 방법 : outb()를 사용하여 이름과 학번을 출력한다.

**ssize\_t dev\_write(struct file \*, const long \*, size\_t, loff\_t \*)**

역할 : application의 데이터를 이용하여 timer를 초기화 시키고 시작 시킨다.

구현 방법 : application에서 넘겨 받은 parameter를 copy from user를 이용하여 kernel level로 복사 한 후, 4 byte stream을 decode하여 timer struct에 입력하고 timer를 등록한다.

**int \_\_init dev\_init(void)**

역할 : fnd driver, led driver, fpga driver와 timer driver를 초기화 한다.

구현 방법 : ioremap를 사용하여 각 device의 address를 mapping 하여 준다.

**void \_\_exit dev\_exit(void)**

역할 : 각 driver의 메모리를 free 시킨다.

구현 방법 : iounmap을 사용하여 mapping 되어 있는 각 device의 address를 free 시킨다.

**3. 시험 및 평가 내용:**

주어진 요구 사항에 대해 만족스러운 output을 만들어 내었다. 여러 차례 다양한 input을 통해 테스트를 돌려 보았고, 별다른 문제를 발견할 수 없었다. 각각의 device 별로 write 함수를 따로 구현하여 문제가 발생한 device에 대해서는 해당하는 부분만 수정할 수 있도록 하였고, application이 수행을 완료하고 rmmod를 사용하여 device driver를 제거할 시 iounmap을 사용하여 메모리를 free 해주기 때문에, 메모리 누수를 막을 수 있었다. 발견되는 문제에 대해 지속적으로 업데이트 하여 계속적인 보완을 할 것이다.

지난 1차 과제에 비해 설계에 비교적 어려움을 겪었다. 다양한 부분을 건드리는 과제였기 때문에 특별한 설계 없이 진행하게 되었지만, device driver를 합치고 같은 input에 맞는 output을 출력하는 과제 였기 때문에 큰 어려움은 없었다.

**V. 기타**

**1. 연구조원 기여도:**

20091648 이준호 : 100%

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

일반적으로 기기 위에서 돌아가는 application을 개발하는 것과 다르게 필요한 모듈을 직접 개발하여 동시에 제어하는 것이 가능하다는 것이 굉장히 흥미롭고 재미있었다. 다만 코드를 여러 파일에 나누어 구현하지 않고 한 파일에 모두 구현한 것이 조금 아쉬웠다. 추가 구현은 ioctl에 대한 이해가 아직 완벽하지 않아 다음 기회로 미루어 둔 것 역시 아쉬운 점이다.