**Embedded 2st HW. Device Driver & Syscall (설계 프로젝트 수행 결과)**

**과목명: 임베디드 시스템 프로그래밍**

**담당교수: 서강대학교 컴퓨터공학과 박 성 용**

**학번 및 이름: 20141542, 심찬양**

**개발기간: 2017. 05. 09. -2017. 05. 11.**

**최 종 보 고 서**

**I. 개발 목표**

이번 프로젝트에서의 목표는 3가지이다. 1. Kernel API라고 불리는 기존의 system call에 추가로 사용자정의의 system call 함수를 작성한다. 2. 여러개의 Device를 제어할 수 있는 Device Driver를 작성하여 module programming 을 한다. 3. 위 2가지의 기능을 모두 사용하는 user application program을 작성한다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

1. 새로운 System call 추가

2. Timer 이용한 Device Driver 개발

3. 1, 2를 종합한 User Application Program 제작

**나. 개발 내용**

1. 새로운 System call 추가

- 프로젝트 명세서에서 요구하는 대로 user application program 에서 argument로 받은 정보를 4byte의 자료형으로 return 하는 system call을 구현한다.

2. timer 를 이용한 Device driver 개발

- linux kernel이 제공하는 jiffies, get\_jiffies\_64() 등을 이용해서 timer interrupt의 interval를 조정하고 주기적으로 특정 handler가 실행되게 한다.

- 이 전 프로젝트에서 사용했던 fpga fnd, led, lcd, dot matrix를 사용하는 device driver를 프로젝트 명세서가 요구하는 대로 작성한다.

3. User Applicaition Program 제작

- 1, 2번을 모두 사용하여 Device driver와 추가된 system call을 활용하는 User Application program을 제작한다.

4. 추가 구현

- Device Driver Moudule 을 개발하고 User Application Program에서 ioctl() 를 이용해서 write() 함수를 호출하여 device driver를 제어 한다.

4. 각 process가 서로 정보를 주고 받을 통신 체계 =>IPC(message queue)

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

**- 5/9 (목) : 지금까지 했던 3주차 system call, 4주차 device\_driver 마지막으로 5주차인 timer interrupt handler 까지 복습하며 프로젝트 명세서를 정독하여 구현하고자 하는 목록을 명세화 했다.**

**- 5/10~11 (금,토) : 3부분으로 나누어진 개발 목표를 system call 부터 추가구현인 ioctl까지 차례대로 구현했다. 또한 Document를 작성하고 마지막 테스트를 마쳤다.**

**나. 개발 환경 및 방법**

**개발 환경 :** Ubuntu Linux 16.04 Terminal에서 vi 편집기를 이용하였으며

arm-none-linux-gnueabi-gcc로 Cross-컴파일을 하였습니다. Adb push 명령을 내 려 usb로 전달한 바이너리 파일을 실행 함으로 결과를 확인 할 수 있었습니다.

**개발 방법 :**

1.System call

- target Board의 kernel에 명세서에서 요구한 새로운 system call을 추가하기 위한 작업을 수행한다. 절차는 아래와 같다 system call number를 할당한다(차례로할당)-> system call table에 새로운 system call을 등록한다. -> kernel의 소스폴더가 있는 곳에 자신이 만든 system call.c 파일을을 저장해 놓는다. -> kernel을 recompile 하여 새로운 bootimage를 만들고 board를 reboot한다. -> User Application program의 실행 파일을 만들어 board에서 실행시켜본다.

2. Device Driver

- host PC에서 device driver module을 생성 한 다음 target board로 옮기는 절차를 수행한다. 절차는 다음과 같다. 먼저 만들고자 하는 device driver의 major number를 결정한다. (겹치지 않는 적절한 번호 설정) device\_dirver.c 파일과 같은 드라이버 코드 파일을 작성하고 make를 하는데 적절한 옵션을 통해 driver.ko 등의 파일을 생성한다. -> board에서 insmod를 수행하여 device driver module를 kernel에 심는 작업을 한다 -> mknod 명령어를 통해 /dev/에 파일을 만들어 user program 과 device driver 사이에서 interface 역할을 수행한다 -> user program을 실행 시켜 device driver가 올바르게 수행되는지 확인한다.

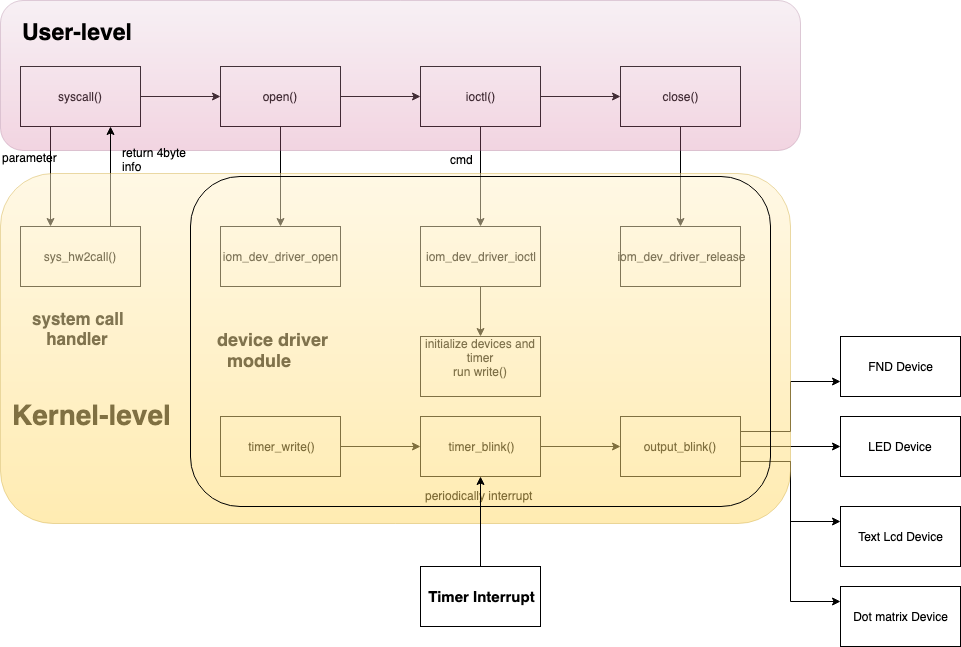
3. user program

- syscall(), open(), write()등의 함수를 사용하여 프로그램을 작성한다. 적절한 parameter를 썼는지 프로젝트 명세서에 요구한 대로 작동하는지 return값을 확인 해 보고 device도 적절히 실행되는지 확인한다.

**IV. 연구 결과**

- 최종 연구 개발 결과를 자유롭게 기술할 것.

**1. 합성 내용:**



**[ 그림 1 ] 전체 프로그램 구성도**

설명 :

1. system call에 의해 Kernel code가 수행되는 경우

- user-level에서 작성한 application program에서 사용한 함수들이 system call을 통해 kernel에 작성된 system call handler와 device driver module 코드가 수행되는 process context를 의미한다.

2. timer interrupt에 의해 kernel code가 수행되는 경우

- ioctl system call을 수행 하여 add\_timer()를 통해 timer list에 새로운 timer를 등록해 놓으면 timer interrupt 가 1ms에 한번씩 interrupt를 걸어서 timer list에 있는 timer 구조체의 expire시간을 확인하여 시간이 만료된 timer list는 종료하고 그렇지 않으면 수행하게 되는 timer interrupt에 의해 Kernel의 timer handler가 수행되는 것은 interrupt context의 프로그램 흐름을 의미한다.

**2. 제작 내용:**

1. system call 개발

- 사용자가 작성한 User Application Program 에서 parameter로 시간 간격, 횟수, 시작옵션을 넘겨 주어 이것들을 종합하여 4byte stream으로 만들어 return 해주는 system call을 작성한다. 이 작업을 위해서 수정한 kernel source는 다음과 같다. ( work/achroimx\_kernel : 커널소스폴더)

1. 커널소스폴더/arch/arm/include/asm/unistd.h

2. 커널소스폴더/arch/arm/kernel/calls.S

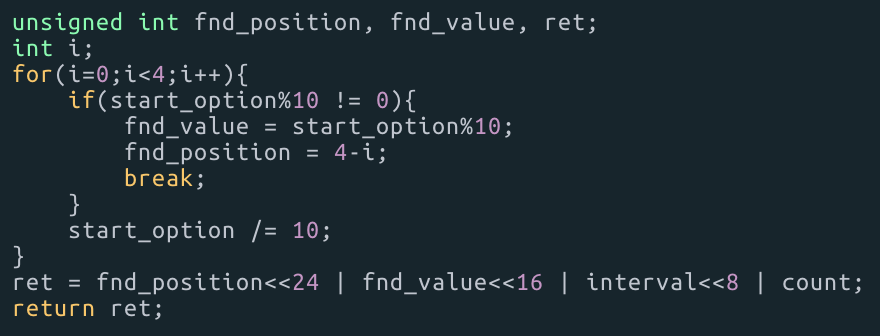
3. 커널소스폴더/include/linux/syscalls.h

4. 커널소스폴더/kernel/hw2call.c

5. 커널소스폴더/kernel/Makefile

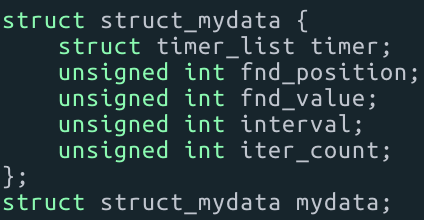
- syscall()를 호출하여 반환 받은 value값은 4바이트 중에 MSB부터 8bit씩 fnd의 위치(1-4), fnd의 값(1-8), 시간 간격 마지막으로 반복 횟수까지 shift 연산을 사용하여 반환 값을 만들었다. 4byte의 unsigned int 형의 자료형을 사용했으므로 처음 1바이트를 <<24 operation을 사용했고 다음은 <<16 <<8 마지막 값은 바로 assign을 하여 총 32bit/ 4byte안에 필요한 정보를 압축하여 저장했다.

Prototype : 

Code : 

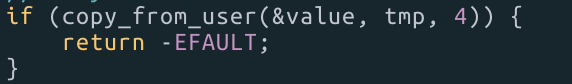
2. device\_driver module 개발

- system call으로 부터 얻은 data를 timer list와 같이 structuring 한 구조체

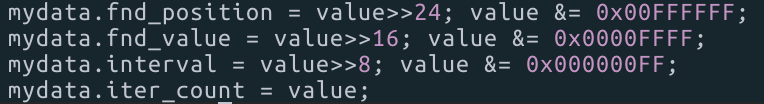


이 data 구조를 사용해서 device driver를 제어했다.

아래의 명령을 사용해서 4byte로 압축된 value값을 받았다.

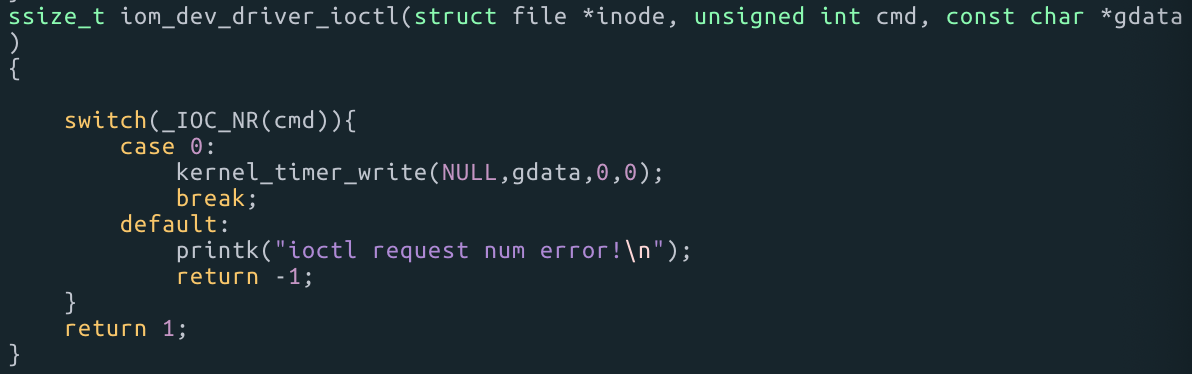


그런 다음 value값 각각 decode하여 struct mydata 에 저장했다.

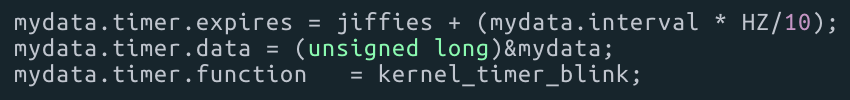


* 응용 프로그램에서 사용한 Ioctl()에서 인자로 넘겨줬던 정보를 device\_driver.c 파일에서 iom\_dev\_driver\_ioctl() 함수의 인자인 cmd 를 decode하여 kernel\_timer\_write()함수를 실행 하였다. 응용 프로그램에서 사용한 \_IOW 는 encoding을 하는 과정이고 \_IOC\_NR()은 magic number를 decode 하는 함수이다.
* #define DEVICE\_MAJOR\_NUM 242
* #define IOCTL\_MY\_WRITE \_IOW(DEVICE\_MAJOR\_NUM,0,4)
* retval=ioctl(dev\_fd,IOCTL\_MY\_WRITE,&ret);

아래는 device\_driver의 코드이다.



Kernel\_timer\_write()가 실행된 후 부터는 timer handler가 1ms 마다 timer interrupt를 발생 시켜 add.timer()를 통해 list에 입력된 timer list timer들의 expire시간을 확인하여 함수를 수행시킬지 종료할지 결정한다.

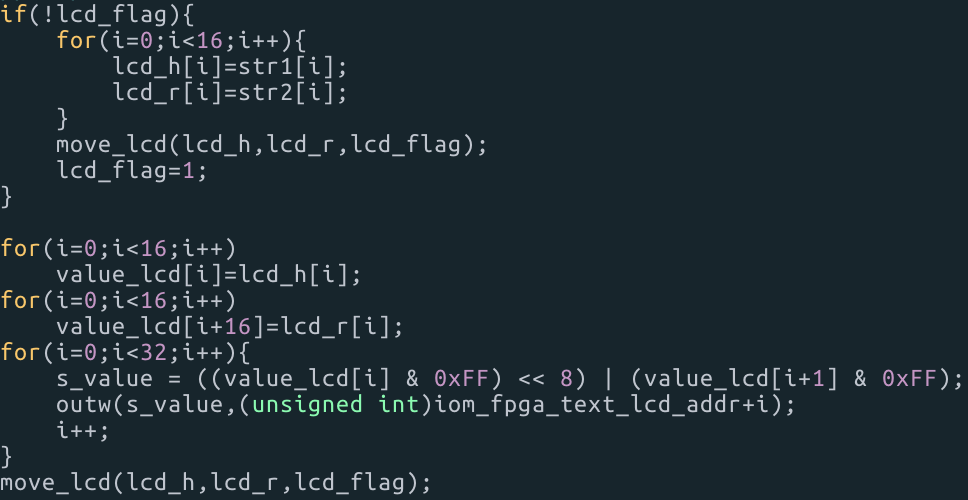


이렇게 expires 시간을 만족하면 kernel\_timer\_blink 함수가 유저가 정한 횟수 만큼 실행하게 된다.

* 실제 device driver를 통해 출력하는 부분

Output\_blink() 라는 함수를 통해서 4가지 device fnd, led, dot, lcd 가 명세서의 요구대로 작동된다.

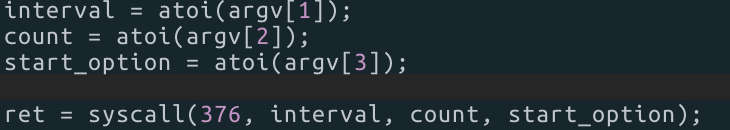
물론 이전에 open() 함수와 module\_init() 통해 device들은 초기화가 되어 있는 상태에서 device를 제어한다. 그 중 lcd device의 코드를 설명하면



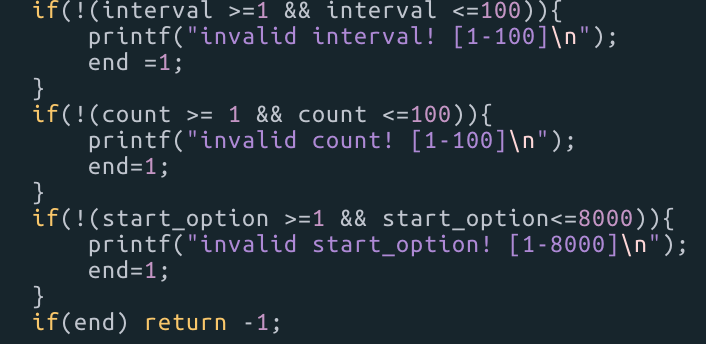
Movd\_lcd()함수를 통해 lcd text가 움직이는 것을 구현했는데 알고리즘은 간단하다 처음은 무조건 오른쪽 방향으로 움직이고 더이상 공간이 없으면 왼쪽으로 움직이게 하는 알고리즘이다. 반대도 마찬가지다.

3. User Application Program

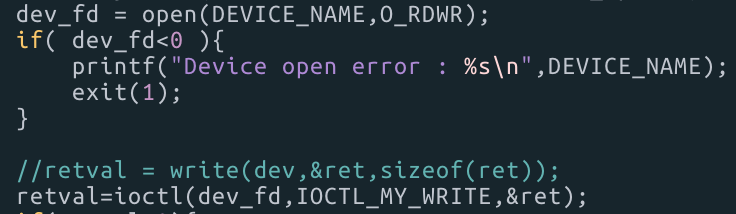
- 사용자로부터 받은 입력 char[]형의 정보를 int형으로 바꾸어 저장하고 그 정보를 1. 절차를 통해 새로 만들어낸 system call을 호출할 때 그 정보를 넘겨 준다.



들어온 값을 확인 해주는 코드

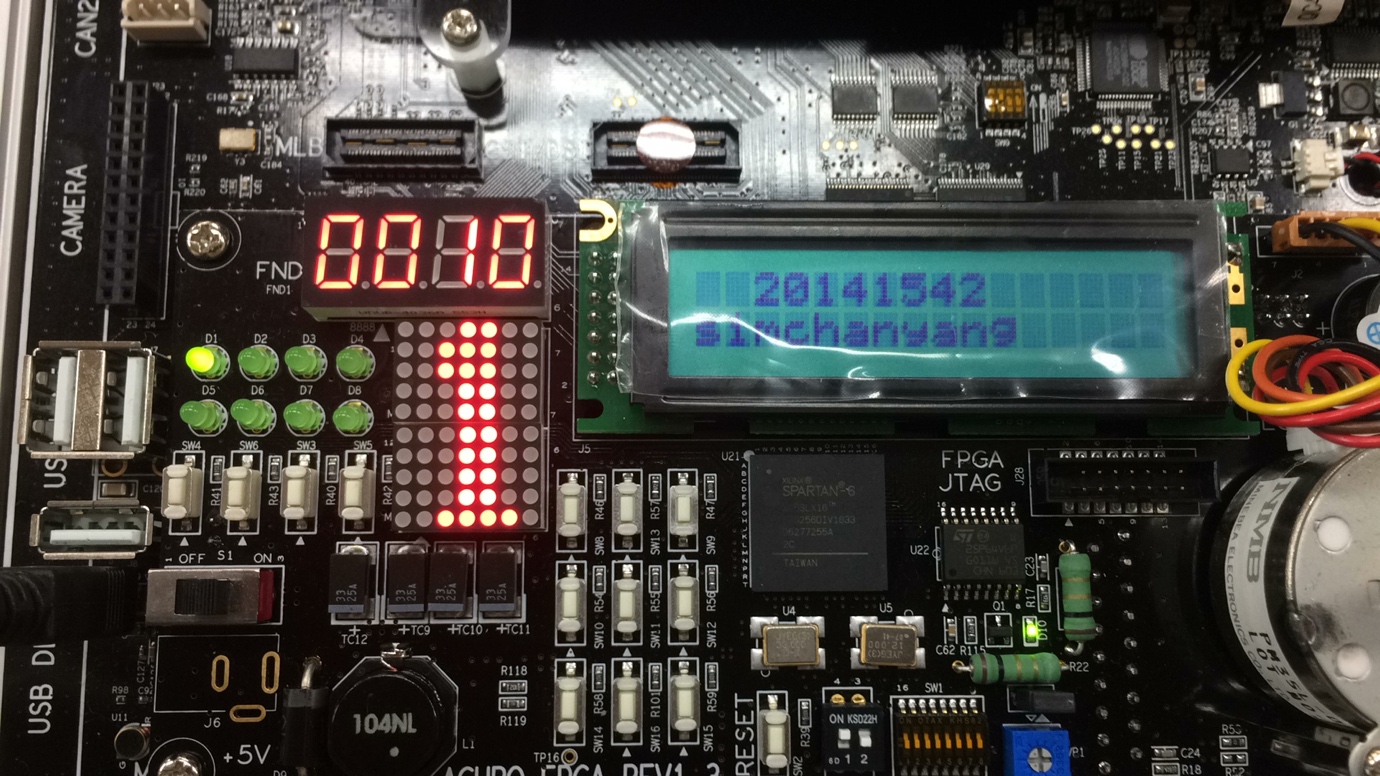


그런 다음 정해진 device\_driver major number와 device\_driver name과 함께 device를 open하고 ioctl() 함수를 통해 Device를 제어 하게 된다.

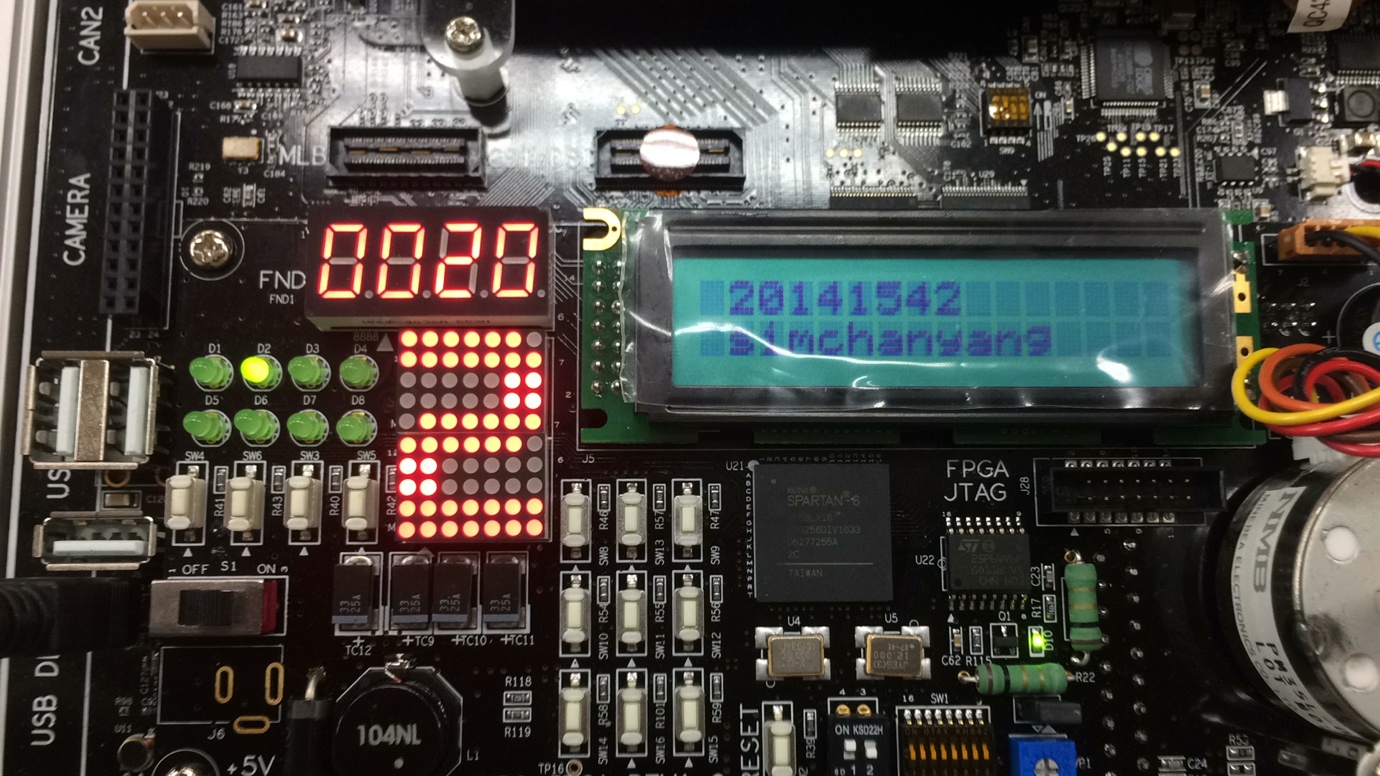


* 실행 예시 화면

Take #1

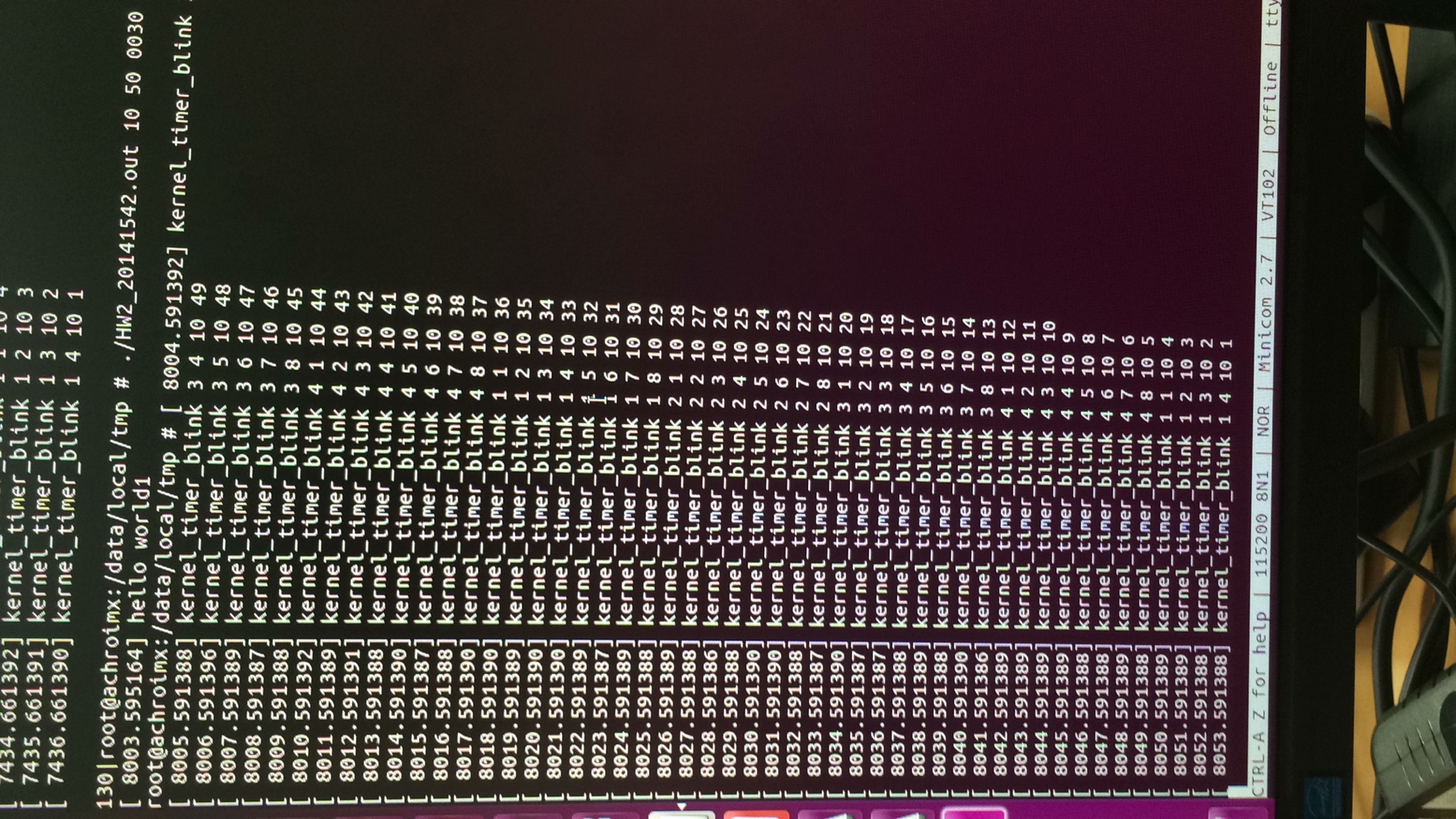


Take #2



설명 : fnd의 숫자와 led(맨앞led 순서대로 1번) , dot matrix 에 나오는 문양이 일치하고 lcd에는 학번과 이름이 오른방향 과 왼방향 한가지 방향으로 움직이는데 한 칸 씩 움직이다 벽에 부딪히게 되면 방향을 바꾸게 된다. 한 칸 씩 움직이는 주기는 사용자가 입력한 interval의 주기 대로 움직이게 된다.

- board 에서 실행 화면



**3. 시험 및 평가 내용:**

- 평가 방법:

명세서에서 요구한 대로 입력을 잘 받는지 User application program에서 미리 확인을 하는 코드를 작성하여 잘못된 입력이 들어오면 프로그램을 종료하였고 올바른 값을 받게 된다면 그 값을 가지고 system call 과 ioctl()을 통한 Device driver control이 올바르게 수행 되는지 몇가지 테스트 케이스를 가지고 검증하였다.

**V. 기타**

**1.** **연구 조원 기여도**:

심찬양 : 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.**

이번 프로젝트는 system call 과 module programming 또한 timer interrupt가 어떻게 이루어 지는지 모두 직접 구현해 볼 수 있었던 점에서 의미가 있었던 것 같다. 앞선 프로젝트에서는 IPC 를 통한 디바이스 제어가 주된 목표였다고 한다면 이번에는 IPC를 하기 전 단계인 kernel에서부터 코드를 고쳐서 사용하는 system call 과 device driver 를 직접 구현해보는 프로젝트 였다. 조금씩 임베디드 시스템의 이해가 넓어지고 있는 것 같다.