**Embedded System Software**

**<Project 2>**

과목명 : 임베디드 시스템 소프트웨어

담당교수 : 서강대학교 컴퓨터공학과 박 성 용

개발자 : 20151607 정재훈

개발기간 : 2019.04.02 ~ 2019.04.15

1. 개발 목표

System call programming, module programming, 디바이스 드라이버 구현 등, 실습 시간 때 배운 내용을 활용하여 프로그램을 작성한다.

1. 개발 내용
2. Device driver

Board의 FPGA device와 timer를 제어하는 하나의 module을 구현한다. 제어해야할 FPGA device는 다음과 같다. FPGA\_FND, FPGA\_LED, FPGA\_DOT, FPGA\_TEXT\_LCD

Device Driver의 기본 기능은 init, exit, open, release, ioctl, timer\_function, fpga\_print 이며 다음과 같은 기능을 수행한다.

* Init

module이 insmod 될 때 수행되는 코드로 Major Number와 Module Name을 kernel에 등록시켜준다. 제어해야할 FPGA device들을 ioremap을 통해 device physical memory를 kernel의 virtual address space에 mapping 시켜준다. 마지막으로 timer list를 초기화 시켜준다.

* exit

module이 rmmod 될 때 수행되는 코드로 kernel에 mapping 시킨 device의 virtual space를 free시키고, timer를 disactivate 시킨다. 마지막으로 등록시킨 device를 등록해제시켜준다.

* open

device 파일을 open시 작동하는 함수로, device는 하나만 open되어 있어야 하므로 이를 check하여 여러번 open될 시 에러를 리턴한다.

* release

device 파일을 close시 작동하는 함수로, 열려 있는 device가 있을 때, device가 더이상 쓰지 않는다는 것을 저장한다. 만약 열려 있는 device가 없다면, 에러를 리턴한다.

* ioctl

device를 control하는 함수로 이번 프로젝트에서는 이를 write대신 사용한다. 응용 프로그램으로부터 4byte로 Encoding된 ioctl type을 받는다. 해당 프로젝트에서 IOCTL로 수행하는 것은 단 한가지로 사용자에게 입력 받은 시간간격[1-100] 횟수[1-100] 시작옵션[0001-8000]을 전달받아, 시간간격에 맞추어 횟수만큼 FPGA device들을 규칙에 맞추어 출력하는 것이다. ioctl type이 알맞게 왔는지 확인한 후, User 영역에서 전달받은 parameter를 통해 시간간격, 횟수, 시작옵션을 도출한다. timer 만료시간, 전달할 data 그리고 timer\_function을 타이머 구조체에 집어넣는다. 마지막으로 이 timer를 timer\_list에 포함시켜 매 timer interrupt 마다 만료시간이 지났는지 확인하고 만료시간이 지났다면 timer\_function을 전달할 data와 함께 호출한다.

* timer\_function

timer\_function은 timer list에 연결된 timer가 exipire될때마다 호출되는 함수로, 총 횟수만큼 호출되도록 한다. 따라서 timer\_function 안에서 호출 횟수를 저장하고, 이를 사용자한테 입력받은 횟수와 비교한다. 만약 횟수만큼 출력되었다면, FPGA device를 초기화 시키고 종료한다. 횟수만큼 출력되지 않았다면, FND에 출력할 다음 number를 계산하고, text lcd buffer를 shift 시킨다. 그 후 timer에 expires, data, function을 넣고 이를 timer\_list에 추가하여 일정시간 후에 다시 timer\_function이 불리게 한다.

* fpga\_print

fpga\_print함수는 현재 저장된 number, 그리고 text\_lcd\_buffer를 참조하여 이를 FPGA devices에 출력해주는 함수이다. FND는 현재 저장된 number를 출력하며 LED는 바뀐 문양의 번호를 출력한다. (D1 : 1번 D2 : 2번 D3 : 3번 D4 : 4번 D5 : 5번 D6 : 6번 D7 : 7번 D8 : 8번) Dot mattrix는 출력중인 문양과 같은 모양의 문양을 출력한다. Text LCD는 text\_lcd\_buffer에 저장된 값을 출력해준다.

1. Parameter들을 받아 하나의 변수로 만들어 return해주는 system call 구현

사용자로부터 시간간격[1-100] 횟수[1-100] 시작옵션[0001-8000]을 입력받고, 이 정보를 저장하는 4byte의 data format이 필요한데, 이를 Encoding하는 기능을 system call로 구현한다. 따라서 parameter는 3개(시간간격, 횟수, 시작옵션)이며 return 값은 해당 포맷으로 인코딩하여 unsigned int data를 반환한다,

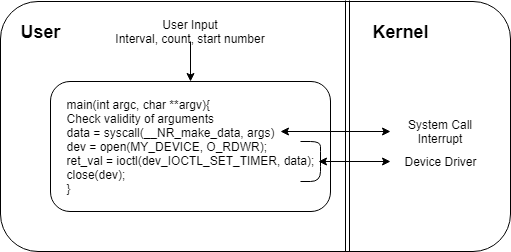
|  |  |  |
| --- | --- | --- |
| 시간간격  1byte | 횟수  1byte | 시작옵션  2byte |

1. Device driver와 System call을 이용하여 출력을 해주는 응용프로그램 구현

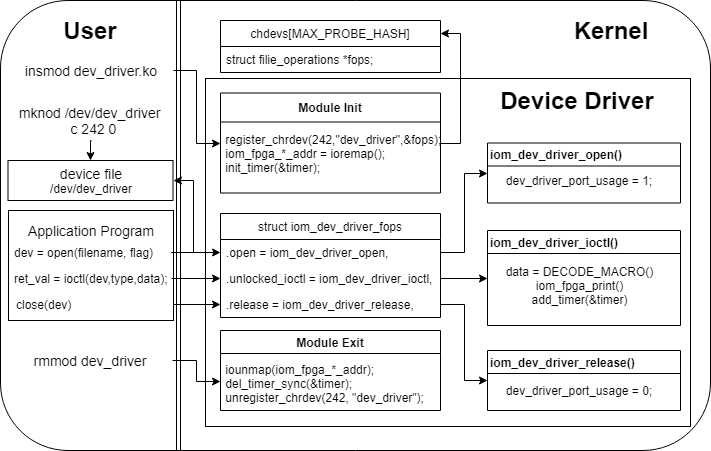
사용자에게 argument 3개를 입력받는다. (시간간격, 횟수, 시작옵션) argument개수가 3개인지, 각 argument range가 적절히 설정되었는지 에러를 체크한 후 System call을 이용하여 이를 4byte data에 Encoding한다. mknod 명령어로 생성된 device file을 open하고, ioctl 함수를 통해 ioctl type과 data를 넘겨준다. 마지막으로 device를 close하고 종료시킨다.

1. 구성

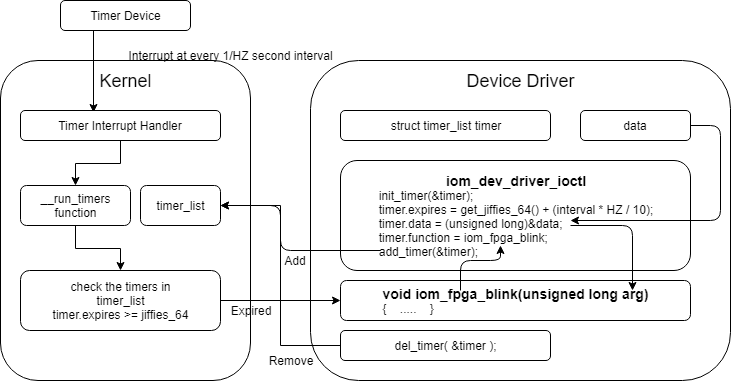
Application Program



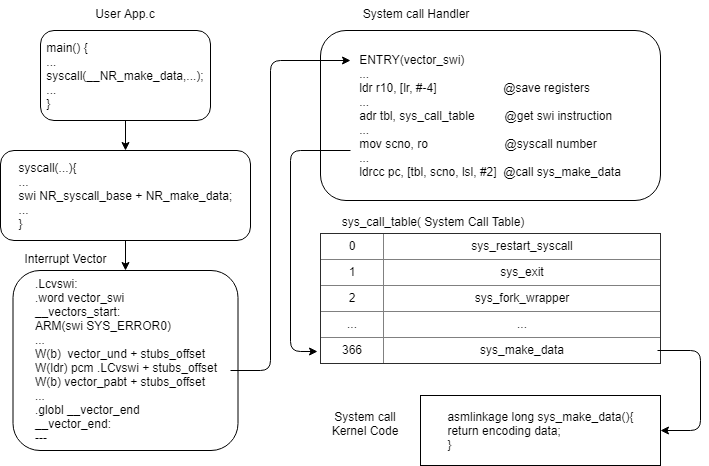
Device Driver Module



Timer Interrupt



System Call Interrupt



1. 제작 내용
2. Device Driver

* init

먼저 register\_chrdev를 통해 Major Number 242와 device name을 Kernel에 등록시킨다. 그 후 ioremap 함수를 이용하여 필요한 FPGA device(Fnd, Text LCD, Dot, LED)들을 kernel address space에 mapping 시킨다. timer 구조체를 초기화 시키고 0을 return한다.

* exit

iounmap function을 통해 mapping 시킨 FPGA device들을 mapping 해제 시켜준다. timer List에 add된 timer 들을 전부 삭제 후 device를 unregister한다.

* open

dev\_driver\_port\_usage라는 변수를 유지하고 open시에 1로 저장한다. 만약 open하려는데 1이 저장된 상태라면 여러번 open하려는 시도로 간주하고 error 를 리턴한다. 또한 open마다 text\_lcd\_buffer에 초기값인 학번, 이름을 setting하고 text\_lcd\_buffer의 shift direction을 setting 한다.

* close

dev\_driver\_port\_usage를 0으로 만들어주며 만약 close 하려는데 이미 0이 저장된 상태라면 여러번 close하려는 시도로 간주하고 error 를 리턴한다.

* ioctl

이 프로그램에서 ioctl type은 하나이며 이는 IOCTL\_SET\_TIMER이다. 이는 매크로로 정의되어있는데 dev\_driver.h에 \_IOR매크로를 이용하였다. type이 IOCTL\_SET\_TIMER이면 ioctl\_param을 parameter로 이용하며 이는 user program에서 Encoding된 data이므로 이를 decode 시켜 interval, count, start\_number를 알아낸다. count는 전역변수로 유지시키며, timer expire을 get\_jiffies\_64() + (interval \* HZ / 10) 로 정하여 interval이 10이라면 1초, 1이라면 0.1초 간격으로 expire 되도록 시킨다. timer를 timer\_list에 집어넣고 FPGA device 초기상황을 print해준다. 매 expire 마다 호출하는 timer\_function으로 iom\_fpga\_blink 함수 포인터를 할당해준다.

* timer\_function (iom\_fpga\_blink)

전역변수로 저장되어 있는 count와 parameter로 전달된 timer관리 구조체의 count를 비교하여 현재 몇번까지 출력되어 있는지 파악한다. 만약 count에 도달했다면 함수를 return 시킨다. 도달하지 않았다면 다음 번에 출력할 number를 계산해준다. 이는 다음과 같은 공식을 따른다

current\_number = (current\_number - i) % ( i \* 8) + i;

(이때 i는 출력중인 자릿수, current\_number가 5000이라면 i = 1000)

current number가 0이 되었을 때 자릿수를 맨 앞으로 옮겨주었다. text\_lcd\_buffer는 현재 shift 방향을 재계산하고 현재 방향에 따라 shift 시켜준다.

right direction을 1로 left direction을 -1로 표현하였고 이에 따른 도달한계 index는 다음과 같다.

(15 \* dir + 15)/2 (dir = 1 -> 15, dir = -1 -> 0 )

이 index에 도달했을시 방향을 바꾸어주며, text\_lcd\_buffer는 총 2줄이므로 2개의 direction 변수를 저장하고 있어야 한다. 관련 변수를 update한 후 iom\_fpga\_print 함수를 통해 현재 저장된 device 변수에 따라 FPGA 기기들을 출력시켜준다. 마지막으로 ioctl 함수와 같이 timer 함수를 timer\_list에 재등록시키고 함수를 종료한다.

* fpga\_print

현재 FPGA 관련 변수 (FND, LED, DOT : int current number, TEXT\_LCD : text\_lcd\_buffer)에 따라 FPGA device를 출력해준다. outw 함수를 통해 mapping된 FPGA device addr에 값을 word size의 값을 전달해준다.

1. System Call

새로운 System Call를 추가하기 위하여 다음과 같은 과정을 거친다.

* Allocation New System Call

arch/arm/include/asm/unistd.h 에 \_\_NR\_make\_data를 356번으로 define한다.

* Registering into System Call Table

arch/arm/kernel/calls.S에 365번 Call 뒤에 CALL(sys\_new\_call)를 추가시킨다.

* Create a New Service Routine

arch/kernel/make\_data.c 라는 파일을 만들고 해당 파일에 system call code를 작성한다. 이때 #include <linux/kernel.h> 를 포함해야한다. 해당 system call은 Encoding하고 return하는 것이므로 (interval << 24) + (count << 16 ) + start\_number를 return한다.

* Expose the prototype of new system call

include/linux/syscalls.h에 make\_data.c에 정의한 sys\_make\_data()를 선언한다.

* Modify Makefile

arch/kernel/Makefile에서 obj-y에 make\_data.o를 뒤에 추가시켜 make시 같이 compile 될 수 있도록 한다.

1. Application Program

유저로부터 받은 argument 개수가 3개인지 파악하고 아니면 error를 리턴한다. 각 argument마다 제약조건이 있는데 interval[1-100] count[1-100] start\_number[0001-8000] 을 지키는지 확인한다. 특히 start number는 4자리로 고정되어야 하며 하나의 숫자만 0이 아닌 숫자여야 한다.

받은 argument를 가지고 위에서 만든 system call make data를 호출하여 encoding된 data를 받는다. open() 함수를 통해 mknod로 생성한 device file을 열고 ioctl(dev, IOCTL\_SET\_TIMER,data)를 호출한다.

(IOCTL\_SET\_TIMER는 ioctl type을 위한 매크로) ioctl이 성공적으로 반환되면 close(dev)를 통해 open 되어 있는 device file을 닫는다.

1. 기타

**느낀점**  
수업시간 때 배운 Device Driver Module, System Call, Timer interrupt를 직접 실습해 보는 경험을 하면서 Kernel에서 Interrupt가 발생되는 과정을 보다 깊게 이해할 수 있었고, Module 형식 프로그래밍의 장점과 단점을 확실하게 깨닫는 계기가 되었다. Timer interrupt 역시 Count 만큼 call되는 함수를 어떻게 짜는지 몰랐는데 project2를 구현하면서 어떤 형식으로 구현해야 간편하고 효율적인지 알게 되었다.