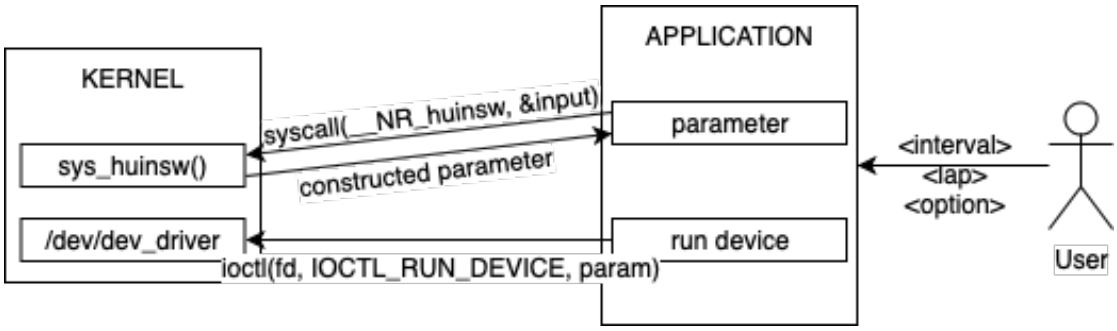
Embedded System Software
HW #2

20151623
한상구

1. 프로젝트 목표

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

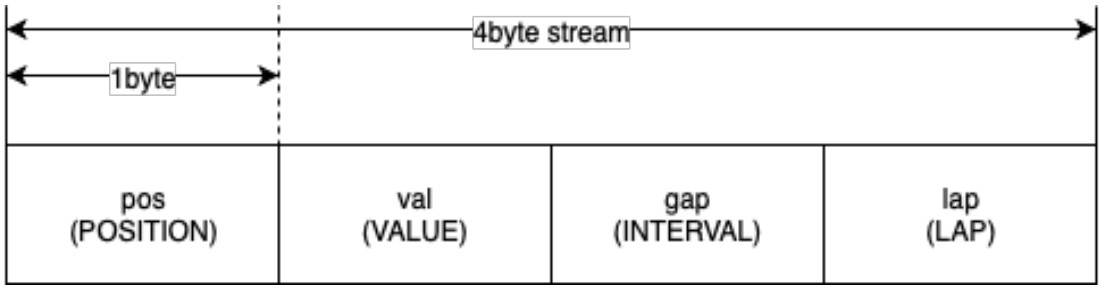
2. 설계



<그림 1 - 전체적인 프로젝트 구조도>

위 그림 1 과 같이 프로젝트를 설계, 구현하였습니다.

Get user input → Construct parameter by using system call → Run device using ioctl 의 흐름이며, 세부 구현 내용은 아래 [3. 기능 구현] 에서 서술하도록 하겠습니다.



<그림 2 - 프로젝트에서 사용되는 parameter 의 구조>

프로젝트 내에서 사용되는 parameter 의 구조는 위 그림 2 와 같이 설계하였습니다.

4byte parameter 에서 각 1byte 단위 component 값을 추출, 4 개의 component 로부터 4byte parameter 를 구축하는 매크로를 작성하여 간편하게 사용할 수 있었습니다.

pos 는 fnd 에서 문양이 출력되는 위치를 담고있으며, 가장 오른쪽 인덱스를 0 으로 설정, 인덱스 값을 저장하고 있습니다. [0, 3] 범위의 값을 갖습니다.

val 은 fnd 에서 출력되는 문양의 정보를 담고있으며, [1, 8] 범위의 값을 갖습니다.

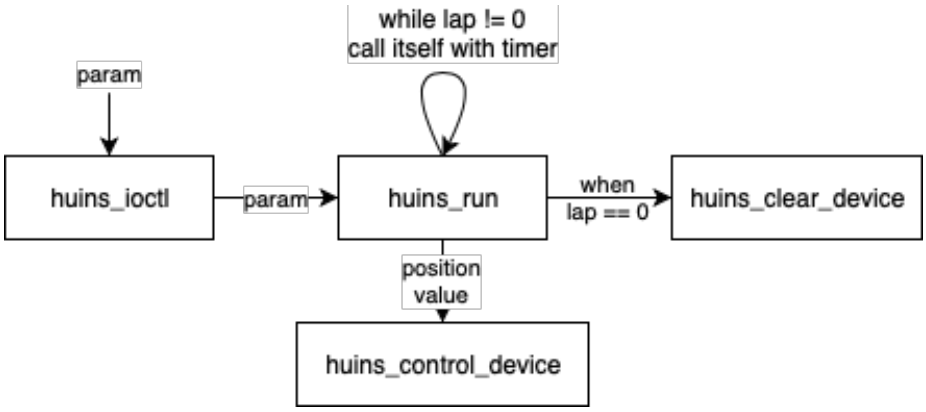
gap 은 출력과 출력 사이의 시간 간격 정보를 담고있으며, [1, 100]의 값을 갖습니다. unit 은 0.1sec 입니다.

lap 은 출력 횟수 정보를 담고있으며, [1, 100]의 값을 갖습니다.

User input 에서 각 component 를 추출하는 과정 및 parameter 를 구축하는 과정은 [3. 기능 구현 - a. Module] 에서 서술하도록 하겠습니다.

3. 기능 구현

a. Module (Device driver, Timer)



<그림 3 - device driver 의 전체적인 흐름도>

전체적인 흐름은 위 그림 3 과 같으며, 위 흐름도에 나오지 않은 함수는 `init_module`, `cleanup_module`, `huins_open`, `huins_release` 로 모듈 및 디바이스 초기화 관련 함수입니다 (보고서에 크게 서술하지 않겠습니다).

`huins_ioctl` 은 application 이 ioctl 을 통해 parameter 를 driver 에 넘겨주면, `copy_from_user` 를 이용해 parameter 를 받아오고 `huins_run` 을 호출하며 parameter 의 주소를 넘겨줍니다.

`huins_run` 은 driver 내에서 사용하는 timer 가 expire 될 때마다 실행되는 함수입니다. 크게 - (i) parameter 에서 component 추출, (ii) device 에 출력, (iii) component 업데이트 및 parameter 재구축, (iv) timer 추가 - 4 가지 파트로 나눌 수 있습니다.

component 추출은 아래 매크로를 통해 구현하였습니다.

```
#define GET_NTH_BYTE(X, N)      (((X) >> ((sizeof(X) - (N)) * 8)) & 0xff)
#define POSITION(X)     (GET_NTH_BYTE(X, 1))
#define VALUE(X)        (GET_NTH_BYTE(X, 2))
#define INTERVAL(X)     (GET_NTH_BYTE(X, 3))
#define LAPS(X)         (GET_NTH_BYTE(X, 4))
```

그림 2 와 같이 parameter 를 구성했기 때문에, parameter 에서 n 번째 byte 를 읽어오는 매크로를 작성, 각 component 를 추출할 수 있었습니다.

device 에 출력하는 파트는 `huins_control_device` 라는 함수를 작성하고 이를 호출하여 진행했습니다.

component 업데이트는 명세서에 따라 진행하였고, parameter 재구축은 아래 매크로를 통해 구현했습니다.

```
#define TO_NTH_BYTE(X, N)       (((X) & 0xff) << ((sizeof(X) - (N)) * 8))
```

```
#define CONSTRUCT_PARAM(P, V, I, L)     ((TO_NTH_BYTE(P, 1)) \
                                        | (TO_NTH_BYTE(V, 2)) \
                                        | (TO_NTH_BYTE(I, 3)) \
                                        | (TO_NTH_BYTE(L, 4)))
```

그림 2 와 같이 구축하기 위해 n 번째 byte 를 그 자리에 맞는 component 로 설정하고, OR 을 통해 재구축을 구현했습니다.

timer 추가는 주어진 옵션인 gap 만큼의 interval 을 주고, 업데이트된 parameter 의 주소값을 data 로 설정한 뒤 `add_timer` 를 통해 구현하였습니다.

`huins_control_device` 는 보드가 출력해야 할 상태를 전달받고, 그에 맞게 보드의 상태를 조정하는 함수입니다. 주어진 옵션을 통해 적절한 state 를 세팅하고, 이를 fpga module address 에 `outw` 를 이용해 출력할 수 있도록 데이터를 쓰는 작업을 진행합니다.

`huins_clear_device` 는 보드의 fpga module 을 전부 초기화 시켜주는 작업을 하는 함수이며, lap 이 0 이 되었을 때 (유저가 입력한 횟수를 모두 채웠을 때) 호출되며 보드를 초기상태로 만들도록 구현했습니다.

b. System Call

Application 이 user input 을 `struct st_huins_op` 에 저장하고, 이 데이터를 위 module 에서 사용할 수 있는 parameter 로 바꾸기 위해 호출하는 system call 을 구현했습니다.

parameter 구축은 위 [a. Module]에서 설명한 `CONSTRUCT_PARAM(P, V, I, L)`을 이용하였고, 주어진 값이 valid 한 경우 구축한 parameter 를, invalid 한 경우 -1 을 반환하도록 했습니다.

c. Application

위에서 서술한 device driver, system call 을 이용하여 device 를 제어할 수 있도록 작성한 응용 프로그램입니다.

User input 을 받고, system call 을 이용하여 device driver 에게 넘길 값을 얻습니다. 이후 ioctl 을 이용하여 device 에 출력을 시작해준 뒤, 종료됩니다.

./app <interval> <lap> <option> 으로 실행할 수 있으며, 각 argument 의 세부 사항은 프로젝트 명세서에서 확인할 수 있습니다.