

Embedded System Term Project

본인이 직접
체크할 것

No	학번	이름
1	201311730	오한진
2	201311734	이건혁
3		
4		
5		

감정 항목			
JNI 에서 배열 덧셈 확인	YES		-30
가변 크기 배열 전달	YES		-20
Buzzer 주기적 On/Off	YES		-10
7-Segment 숫자 움직임	YES		-20
감점 합계			

(소스 코드와 다를 경우 항목별 추가적으로 -10점 감점)

항목	Points
S/W 요구 사양서 (20)	
S/W 설계서 (20)	
프로그램 빌드 과정 (10)	
정상 동작 여부 (50)	
Total	

S/W 요구 사양서

/* 디바이스 드라이버의 요구 사항 및 가정 사항에 대해서 상세히 기술 한다. */

/* 예를 들어, 디바이스 드라이버의 기본 동작 및 예외 현상 발생 시 동작에 대한 요구 사항 등을 기술하도록 한다. */

1. Buzzer 드라이버

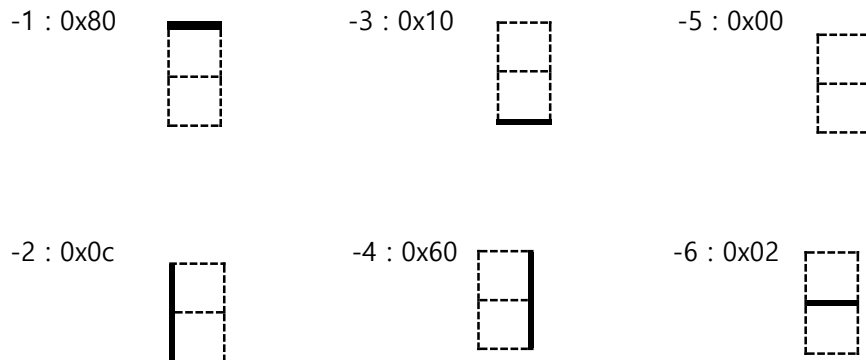
Buzzer 드라이버는 기본적으로 신호가 주어진다면 소리가 나게끔 하는 일이 전부이다. 끄고 켜고 하거나 반복적으로 소리를 내는 기능은 Activity 에서 구현했다.

그렇기 때문에, 기본적인 open, write, release 기능과 init 함수와 exit 함수를 가지고 있는 간단한 형태이다.

2. 7-Segment 드라이버

7-Segment 의 경우, 기본적인 기능은 Activity 에서 JNI 함수인 SegmentControl(int) 함수를 이용해 호출을 하고 입력된 값만큼 화면에 띄우는 드라이버이다. 여기서 우리는 Segment 화면을 벗어나거나, 입력할 필요가 없는 수(0~999999 를 제외한 수들)를 활용해 새로운 Segment 화면을 넣었다.

write 함수에서 Getsegmentcode(short)함수를 이용해 각 필요한 모양의 값들을 불러와 data 에 넣는 방식인데, 우리는 Getsegmentcode 안의 switch 에 0~9 까지의 숫자 모양 외에 -1~-6 까지의 음수를 추가했다. 그리고 각 숫자에 임의로 만든 모양의 segment 코드를 넣었다.



그래서 만약 앱 쪽에서 대기 모양이 필요하면 -1 를 각 digit 칸마다 6 번 반복 후, -2, -3, -4 로 이동함으로써 대기 화면을 만들었다. SegmentContrl 을 통해 앱에서 호출 할 때, 들어오는 매개변수 값의 범위를 통해 APP 이 원하는 출력을 하게끔 만들었다. 이 때, if~else if 문의 모양을 통해 만들었다.

if(매개변수값이 0 보다 작을 때) : 음수의 값이 들어오면, 대기값 출력으로 받아들이고, 사용자가 지정한 모양에 따라 data 값을 설정했다. 예를 들면, SegmentControl(-155555)를 실행시키면, 맨 왼쪽 위에만 - 자 모양이 나오고 나머지는 다 비어있는 모양이다.

else if(데이터값이 1000000 보다 크고 200000 보다 작은경우) : 정답의 수를 출력하기 위함이다. 이때 20000xx 와 같은 숫자가 들어오는데, xx 만 따로 분리해 쓰는 기능으로 구현했다. 이는 200 만을 나눈 나머지를 갖고오면 xx 의 숫자만 들어온다. 그리고 이 안에서 또 2 가지의 경우로 나뉘어진다. **1. 정답의 값이 1 의 자리만 있을 때, 2. 정답의 값이 10 의 자리일 때** 이다. 그리고 수가 움직이는 원리는 index 를 사용했다. 출력 후, index 의 값을 상승시켜서 다음에 또 출력 할 때, index 의 값에 따라 위치를 조정하는 원리이다. 이 때, index 값을 하나만 쓰면, case 를 0, 200, 400..순으로 올라가야 하는데 (왜냐하면 한번만 수행되는게 아닌, 몇 십 ~ 몇 백번까지 반복되는데, 한번만 실행되고 index 가 200 번까지 올라가는 동안 빛이 안나오기 때문에, index2 를 만들어 index2 가 오르면서, index2 가 200 단위에 다다르면 그때야 index 을 1 씩 더해 200 번씩 많은 출력을 함에도 지속적으로 수행되게끔 만들었다.

else if(데이터의 값이 2000000 보다 크고, 3000000 보다 작을 때) : SegmentControl(2xx00xx)와 같은 모양으로 사용하게끔 만들었다. 여기서 위와 같은 방법으로 2xx00xx 중 앞 사용자가 입력한 xx 와 뒤의 정답 값인 xx 를 분리해서 사용한다. 앞의 xx 는 10000 으로 나눈 몫을 가져오고 뒤의 xx 는 나눈 나머지를 가져오면 된다. 출력은 정답때와 같은 원리의 index, index2 를 사용했으며, 가운데 -모양의 Getsegmentcode(-6)을 넣어서 출력하게끔 만들었다.

else if(데이터의 값이 3 백만일 때) : index, index2 를 초기화 한다. 이 기능을 넣은 이유는, 만약 숫자가 segment 에 출력하는 도중 clear 나 다른 수를 넣고 ok 를 눌러버리면 index 가 초기화되지 않고 전에 갖고 있던 값 그대로 유지되어서 다음 segment 출력시 위치가 꼬이는 문제가 발생한다. 이를 의도적으로 초기화 하기 위해 넣었다.

else if(0 보다 크고 1 백만보다 작은 경우) : 일반 segment 가 출력되는 경우이다. ArrayAdder 앱의 경우 이를 사용하는 경우는 없다.

S/W 설계서

/* 작성한 S/W 의 전체 구조, 각 함수별 동작 방법 등 구현한 S/W 에 대한 상세 설명 기술 */

1. 안드로이드

1-1 XML 부분

전체적인 어플리케이션 화면 전체를 RelativeLayout 을 이용해 감싸놓고 그 이후 아래의 모든 레이아웃들은 LinearLayout 을 이용해 작성하였습니다. 맨 위의 배열을 생성하는 부분은 spinner 를 이용하여 미리 생성했던 array.xml 파일 내의 item 속성값을 저장하고 이를 사용자가 선택하여 원하는 배열의 인덱스값을 지정할 수 있도록 만들었습니다.

배열 생성 후의 각 Layout 들은 배열 출력(vision1), 정답 입력(vision1_1), 정답 여부 출력(vision2), 확인과 초기화 버튼 레이아웃(vision1_2)로 id 를 설정한 후 초기 visible 속성을 gone 으로 하여 배열 생성 버튼이 눌리기 전까지는 사용자의 눈에 드러나지 않도록 하였습니다.

vision1 에는 배열 출력을 위한 TextView(arrayResult)를 배치하였습니다.

vision2 에는 정답 입력을 위한 EditText(inputNum)을 배치하였으며 사용자의 이해가 쉽도록 hint 속성을 부여하였습니다.

vision1_1 에는 정답 여부를 출력할 수 있는 TextView(yesOrNo)를 배치하였습니다.

vision1_2 에는 확인과 초기화를 위한 Button 두 개(okButton, clearButton)를 배치하였으며 둘의 비율을 같게 하기 위해 layout_weight 속성값을 1 로 하였습니다.

1-2 Activity 부분

1-2-1 배열 생성

문제에서 요구하는 배열을 생성하기 위하여 spinner 를 앞선 xml 에서 배치하였고, 이를 activity 내에서 사용하기 위해 ArrayAdapter 를 생성하여 array.xml 의 속성값들을 spinner 와 결합시켰습니다. 다음으로 배열의 값이 출력될 Layout(vision1)도 생성하고 레이아웃의 속성값을 변화시키기 위한 레이아웃 파라미터 객체 param 도 선언해주며, 배열이 생성된 후 인덱스 별 랜덤한 값이 어떻게 배치되었는지에 대한 결과를 출력하기 위해 TextView(arrayResult)를 만들고 이 곳에 출력될 ArrayList(list)도 함께 선언해주며, 이 View 가 포함된 Layout 의 높이가 고정되지 않고 index 의 크기에 따라 유동적으로 변화하게 만들어 줄 int 형 변수인 lengthResult 도 설정해줍니다. 배열 생성을 위한 Button(arrayCreate)을 생성한 후에는 이 버튼에 클릭되었을 때의 Listener 를 설정하게 됩니다.

리스너 내부에는 onClick 메소드가 존재하며 배열이 눌렸을 때마다 vision1 에서 출력될 내용이 계속 바뀌어야 함으로 list 변수를 clear 시키고 arrayResult 에 ""를 setText 시켜 깨끗하게 비워줍니다. int 형 변수인 number 를 생성하여 spinner 의 속성값 중 사용자가 선택한 값을 저장해준 다음 이 값을 미리 불러왔던 JNI 함수인 CreateArray 에 넣어 디바이스 드라이버에서 인덱스별 무작위 값을 설정하여 배열을 다시 돌려줄 수 있도록 합니다. 이 후 결과값을 int 형 배열인 sumArray 에

넣어줍니다. 이후 lengthResult 의 값을 0 으로 초기화시켜 매번 배열 생성버튼이 눌리더라도 중복 되어 길이가 늘어나지 않도록 만듭니다.

받아온 배열의 길이만큼 list 에 출력할 문장을 저장하며 lengthResult 의 값도 늘어나도록 하고, 이 값을 param 객체의 height 에 설정 후 vision 에 set 시켜줍니다. 반복이 끝나게 되면 Iterator 을 이용하여 list 에 저장되었던 값들을 순서대로 arrayResult 에 넣어줍니다.

1-2-2 정답 입력 및 확인

배열 생성 버튼이 눌렸을 때 드러나게 되는 EditText(inputNum)에 사용자가 값을 입력한 후 확인 버튼을 누르면 Button(오케이)의 리스너가 실행됩니다. 사용자가 입력한 값을 받아 저장하는 int 형 변수인 sum 은 JNI 함수인 CompareArray 에 sumArray 와 함께 전달되어 인덱스들의 총 합과 사용자가 입력한 정답을 비교합니다. 이후 정답일 경우 1, 오답일 경우 0 을 리턴시켜 result 에 저장합니다. 또한 해당 리턴값을 부저 울림에 이용할 int 형 변수 BuzData 에 저장합니다. 정답일 때는 시간에 따라 부저가 울렸다 울리지 않았을 수 있어야 합니다. 그렇기 때문에 TimerTask 를 이용하여 시간에 따라 BuzData 가 1 과 0 을 반복시킬 수 있도록 설정시킨 다음, JNI 의 CompareArray 가 1 을 리턴하면 timertask 로부터 schedule 를 사용해 스위치되는 시간의 간격을 설정하며 그 후 BuzzerControl 함수를 실행시켜 BuzData 값에 1 과 0 을 반복 저장함으로써 부저가 간격을 두며 울리도록 합니다. 리턴 값이 0 일 경우에는 TimerTask 를 거칠 필요가 없으니 BuzData 를 1 로 바꾼 후 그대로 BuzzerControl 함수를 실행시켜 부저를 쭉 울리도록 만듭니다.

1-2-3 정답여부 출력

vision1_1 로서 배열 생성과 함께 드러나는 이 부분은 초기의 inputNum 의 값이 null 이면 “아무것도 입력이 안되었습니다.”라는 말을 TextView(yesOrNo)에 저장하며, 100 이상의 값을 입력했을 경우 “1~99 까지의 값을 입력하세요.”를 출력하도록 하였습니다. 이들 예외시에는 부저가 울리면 안되기 때문에 BuzData 를 0 으로 설정하여 올바른 답을 입력한 후 예외가 발생하였을 때도 부저가 계속 울리지 않도록 하였습니다. 예외를 제외한 정상적인 경우에는 CompareArray 로부터 저장한 값인 result 가 1 일 때는 yesOrNo 에 “정확합니다”를, 0 일 때는 “틀렸습니다.”를 set 하도록 하였습니다.

1-2-4 초기화

Clear 버튼을 입력하게 될 경우 모든 것이 초기화되어야 하여 BuzData 를 0 으로 변경한 후 BuzzerControl 함수를 실행시켜 울리던 부저가 멈추게 하며, 입력한 배열 인덱스에 따라 print 를 수행시켜주던 레이아웃의 크기도 다시 초기화시켜주기 위해 lengthResult 또한 0 으로 변경시켜주고, 마지막으로 배열 생성 레이아웃을 제외한 모든 레이아웃들의 visibility 옵션을 gone 으로 변경해주어 다시금 화면에 드러나지 않도록 만듭니다.

1-2-5 segment 부분

위에서 정답이 맞을 때, 틀렸을 때 flag 값이 바뀌면서 아래서 만든 Thread 에서 segment 가 실행되는 구조로 만들었다. Flag 가 -1 일 때는 기본적인 대기모양의 segment 를 출력하게끔 만들었다.

대기 모양을 출력하기 위한 함수 `repeatSegmentControl_Box()`를 만들어서 사용한다. 대기상태 일 때의 -1~-5 까지의 data 값을 활용해서 마치 네모난 박스가 보이게끔 만들었다.

`SegmentControl(-211114);` 맨 왼쪽과 오른쪽엔 | 모양이 나오고 나머지는 위의 - 모양이 출력

`SegmentControl(-233334);` 맨 왼쪽과 오른쪽엔 | 모양이 나오고 나머지는 아래의 - 모양이 출력

`SegmentControl(-111111);` 모두 위의 - 모양이 출력

`SegmentControl(-333333);` 모두 아래의 - 모양이 출력

나머지 출력들은 200 번씩 반복해 출력함으로써 계속해서 유지되면서 출력되어 있는것처럼 보이 기 위한 함수 `repeatSegmentControl` 를 사용했다.

`repeatSegmentControl` 을 호출하면서 매개변수에 따라 사용자가 사용하고자 하는 segment 모양 이 출력된다. (이는 드라이버에서 분류해서 출력하게끔 만들었고, Activity 에서는 원하는 모양대로 간편하게 매개변수만 잘 넣어서 보내면 된다.)

2. JNI

JNI 에서는 총 5 가지의 메소드가 구현되어 있다.

`Jint Java_ac_kr_kgu_esproject_ArrayAdderActivity_BuzzerControl(JNIEnv* env, jobject thiz, jint value)`

→ 부저를 컨트롤 하기 위한 메소드, value 값이 1 이면 소리가나고 0 이면 안난다.

`Jint Java_ac_kr_kgu_esproject_ArrayAdderActivity_SegmentControl(JNIEnv* env, jobject thiz, jint data)`

→ 세그먼트를 컨트롤 하기 위한 메소드, data 값에 따라 출력모양이 바뀐다. (드라이버에서 구현)

`Jint Java_ac_kr_kgu_esproject_ArrayAdderActivity_SegmentIOContrl(JNIEnv* env, jobject thiz, jint data)`

→ 세그먼트출력의 유형을 지정하기 위한 메소드, `ArrayAdderActivity` 에서는 0 으로 고정되어 있 다. (0 일시 일반출력, 1 일시 시간출력 : . 이 추가루 찍힘)

`jintArray Java_ac_kr_kgu_esproject_ArrayAdderActivity_CreateArray(JNIEnv* env, jobject thiz, jint value)`

→ 들어온 value 값의 길이만큼 배열을 생성해서 배열 그대로 return 하는 메소드. 이를 이용해 가변적인 길이를 가진 배열을 받을 수 있다.

`jint Java_ac_kr_kgu_esproject_ArrayAdderActivity_CompareArray(JNIEnv* env, jobject thiz, jint value, jintArray sumArray)`

→ 배열과 value 값을 받아 배열안의 값과 value 를 비교해 합이 같은지 틀린지를 비교해 맞을때 는 1, 틀릴때는 0 을 return 하는 메소드

프로그램 빌드 과정

/* Android 프로그램, JNI 프로그램 및 디바이스 드라이버 생성 과정 상세 기술 및 화면 캡처 */
/* 기술한 형태로 프로그램이 정상적으로 생성되지 않을 경우 비정상 동작으로 간주함 */

1. 드라이버 생성(segment.ko, buzzer.ko)

컴파일은 /Android/drivers/각 드라이버 폴더/ 에서 수행한다.

```
hanback@hanback: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
  
root@hanback:/# cd /Android/drivers/248.buzzer_driver/  
root@hanback:/Android/drivers/248.buzzer_driver# make  
make -C /Android/linux-2.6.32-hanback SUBDIRS=/Android/drivers/248.buzzer_driver modules  
make[1]: Entering directory `/Android/linux-2.6.32-hanback'  
Building modules, stage 2.  
MODPOST 1 modules  
make[1]: Leaving directory `/Android/linux-2.6.32-hanback'  
rm -f default  
root@hanback:/Android/drivers/248.buzzer_driver#
```

```
hanback@hanback: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
  
root@hanback:/# cd /Android/drivers/242.seg_driver/  
root@hanback:/Android/drivers/242.seg_driver# make  
make -C /Android/linux-2.6.32-hanback SUBDIRS=/Android/drivers/242.seg_driver modules  
make[1]: Entering directory `/Android/linux-2.6.32-hanback'  
Building modules, stage 2.  
MODPOST 1 modules  
make[1]: Leaving directory `/Android/linux-2.6.32-hanback'  
rm -f default  
root@hanback:/Android/drivers/242.seg_driver#
```

2. JNI (ESTermProject.so)


컴파일은 /Android/android-ndk-r5b 에서 수행하며, make APP=ESTermProject 명령어를 사용

```
hanback@hanback: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
  
root@hanback:/Android/android-ndk-r5b# make APP=ESTermProject  
Android NDK: Building for application 'ESTermProject'  
Compile thumb : ESTermProject <= ArrayAdder.c  
SharedLibrary : libESTermProject.so  
Install : libESTermProject.so => apps/ESTermProject/project/libs/armeabi/libESTermProject.so  
root@hanback:/Android/android-ndk-r5b#
```

3. 드라이버를 타겟머신에 적재

명령프롬프트에서 adb shell 을 이용해 /modules/fpga 에 buzzer.ko 와 segment.ko 파일을 적재한다.

앞서 적재했던 ko 파일들을 rmmod 명령어를 이용해 지우고 시작한다.

 명령 프롬프트 - adb shell

```
Microsoft Windows [Version 10.0.17134.407]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Wohhan>adb shell
# cd /modules/fpga
cd /modules/fpga
# rmmod segment.ko
rmmod segment.ko
# rmmod buzzer.ko
rmmod buzzer.ko
# exit
exit

C:\Users\Wohhan>cd C:\work\WES

C:\work\WES>adb push segment.ko /modules/fpga
696 KB/s (67025 bytes in 0.093s)

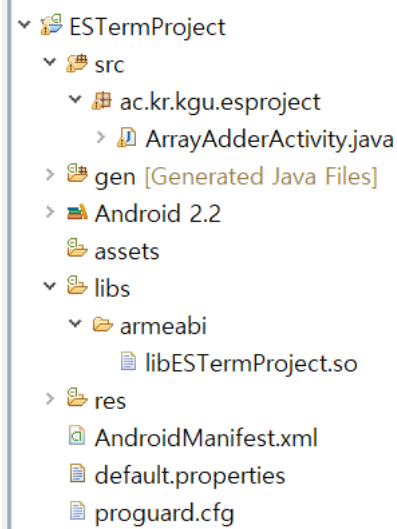
C:\work\WES>adb push buzzer.ko /modules/fpga
628 KB/s (60519 bytes in 0.093s)

C:\work\WES>adbshell
'adbshell'은(는) 내부 또는 외부 명령, 실행할 수 있는 배치
파일이 아닙니다.

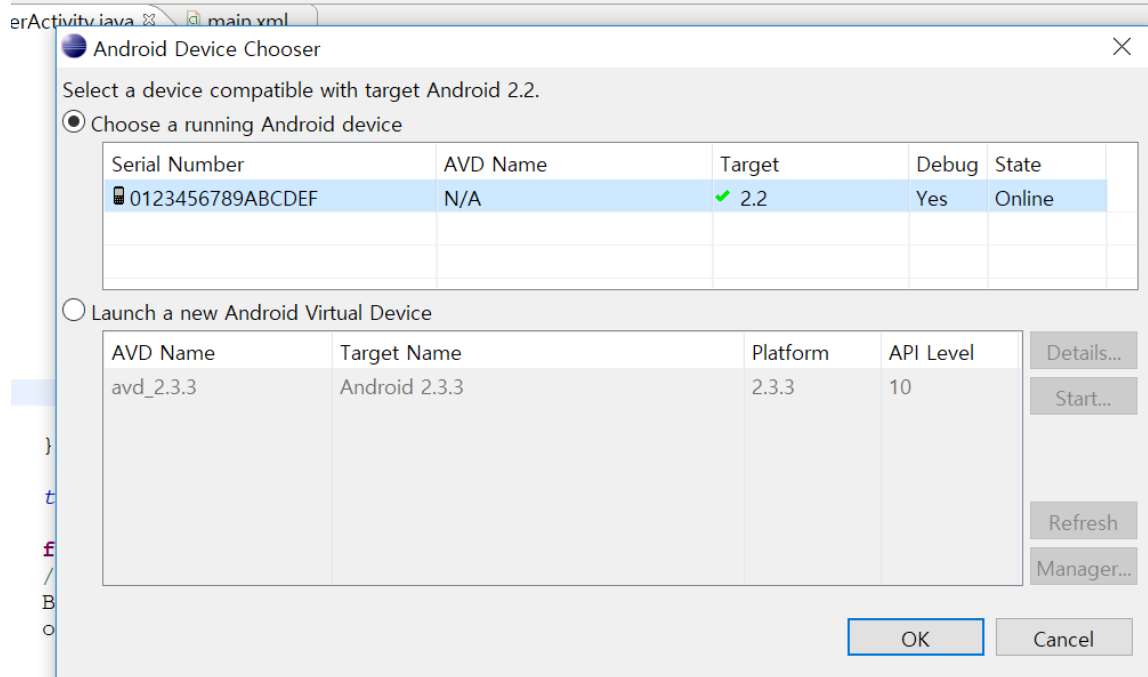
C:\work\WES>adb shell
# cd /modules/fpga
cd /modules/fpga
# insmod segment.ko
insmod segment.ko
# insmod buzzer.ko
insmod buzzer.ko
#
```

4. 이클립스에서의 안드로이드 앱 빌드

이클립스에서의 프로젝트의 모양은 다음과 같다. JNI 로 만들었던 libESTermProject.so 파일을 하위폴더 libs/armeabi/ 폴더에 넣는다.



그 후, 이클립스에서 타겟머신을 지정해 실행한다.



5. 타겟머신 화면

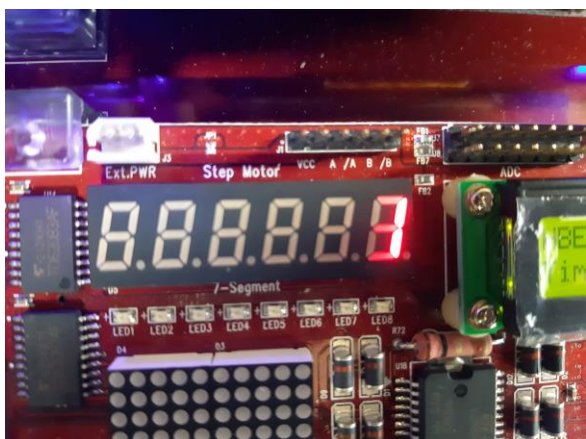
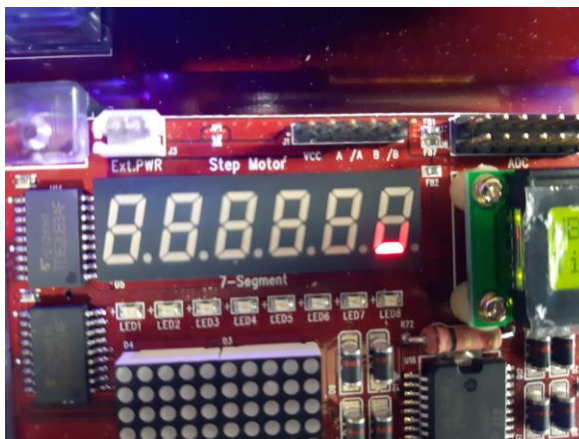
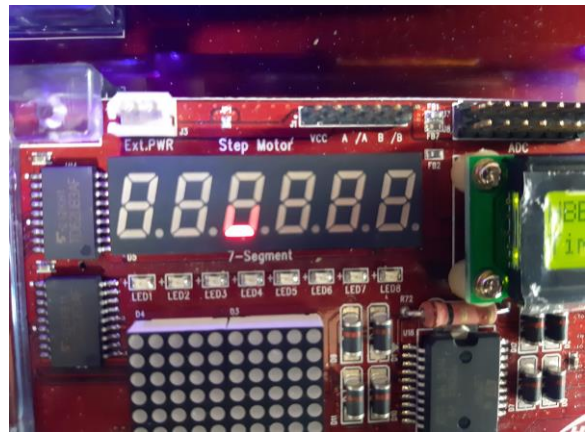
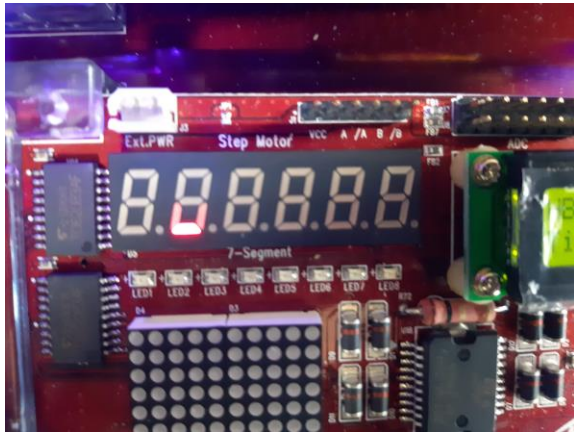
앱 화면 작동/부저 작동

➔ 동영상 첨부

맞췄을 시, 삐 - 삐 - 삐 - 일정하게 반복되고, 틀렸을 시 삐----- 일정한 소리가 난다.

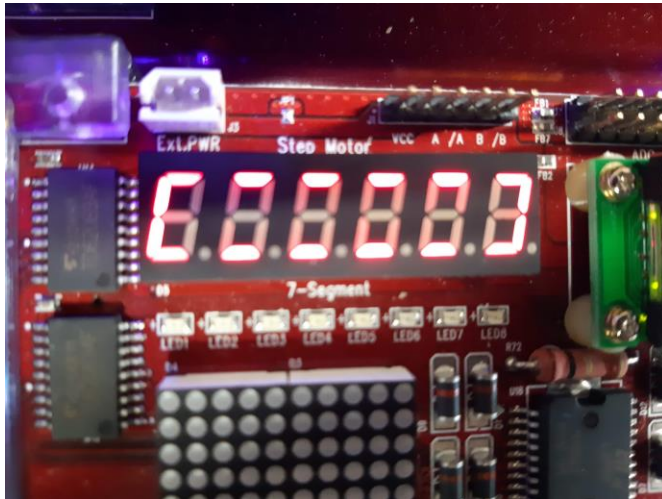
Segment 화면

-일반 대기일 때



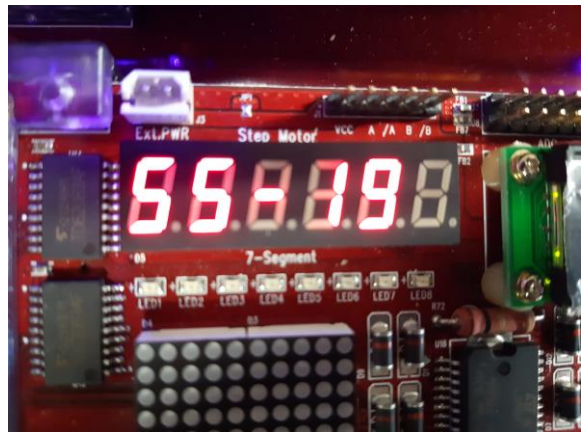
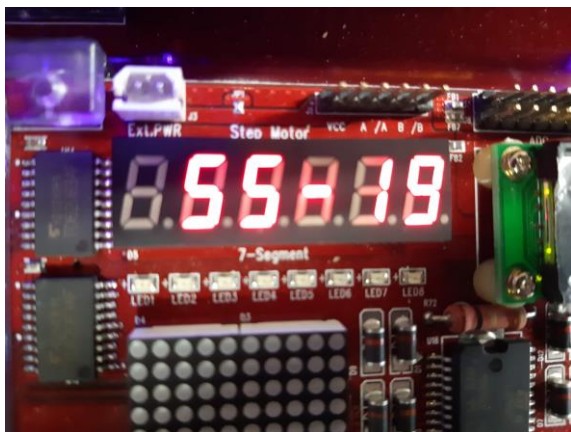
위 사진과 같이 계속해서 테두리를 빙글빙글 돌게끔 함
Clear 버튼을 눌렀을 때도 다시 위와 같은 대기상태로 복귀

-정답 or 오답을 출력할 때



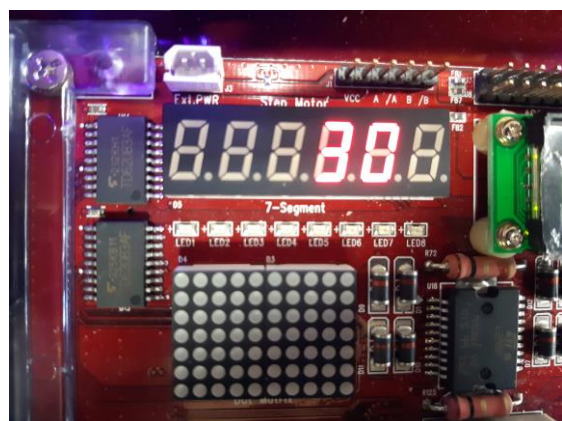
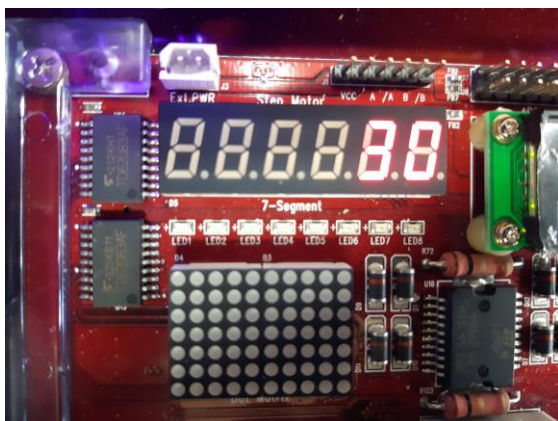
이와 같은 직사각형 모양이 일정하게 3 번 빛난 후 오답, 정답이 출력됨
그리고 출력이 끝난후 직사각형 모양이 또다시 3 번 반복해서 나온 후 정답을 출력하는 동작을
계속해서 반복함

-오답일 시



위와 같은 사용자 입력값 - 정답 모양의 Segment 가 한칸씩 왼쪽으로 움직여서 사라질때까지 움직임

-정답일 시



정답일 때도 마찬가지로 한칸씩 왼쪽으로 움직임

6. Source 코드

(1) 디바이스 드라이버 (buzzer) (/Android/drivers/248.buzzer_driver/)

Buzzer.c 파일

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/fs.h>
#include <asm/io.h>
#include <asm/uaccess.h>

#define DRIVER_AUTHOR "hanback"
#define DRIVER_DESC "buzzer test program"
#define BUZZER_MAJOR 248
#define BUZZER_NAME "BUZZOR IO PORT"
#define BUZZER_MODULE_VERSION "BUZZER IO PORT V0.1"
#define BUZZER_ADDRESS 0x88000050
#define BUZZER_ADDRESS_RANGE 0x1000

//Global variable
static int buzzer_usage = 0;
static unsigned long *buzzer_ioremap;

//define function...
//응용 프로그램에서 디바이스를 처음 사용하는 경우를 처리하는 함수
int buzzer_open(struct inode *minode, struct file *mfile)
{
    //디바이스가 열려있는지 확인
    if(buzzer_usage != 0) return -EBUSY;
    //buzzer 의 가상 주소 매핑
    buzzer_ioremap = ioremap(BUZZER_ADDRESS, BUZZER_ADDRESS_RANGE);
    //등록할 수 있는 I/O 영역인지 확인
    if(!check_mem_region((unsigned long)buzzer_ioremap, BUZZER_ADDRESS_RANGE)){
        //I/O 메모리 영역을 등록
        request_mem_region((unsigned long)buzzer_ioremap,
BUZZER_ADDRESS_RANGE, BUZZER_NAME);
    }
    else
        printk(KERN_WARNING"Can't get I/O Region 0x%xWn", (unsigned
int)buzzer_ioremap);
```

```

        buzzer_usage = 1;
        return 0;
    }
    //응용 프로그램에서 디바이스를 더이상 사용하지 않아서 닫기를 구현하는 경우
    int buzzer_release(struct inode *minode, struct file *mfile)
    {
        //매핑된 가상주소를 해제
        iounmap(buzzer_ioremap);
        //등록된 I/O 메모리 영역을 해제
        release_mem_region((unsigned long)buzzer_ioremap, BUZZER_ADDRESS_RANGE);
        buzzer_usage = 0;
        return 0;
    }
    //디바이스 드라이버의 쓰기를 구현하는 함수
    ssize_t buzzer_write_byte(struct file *inode, const char *gdata, size_t length, loff_t *off_what)
    {
        unsigned char *addr;
        unsigned char c;
        //gdata의 사용자 공간의 메모리에서 c에 읽어온다.
        get_user(c,gdata);
        addr = (unsigned char *)(buzzer_ioremap);
        *addr = c;
        return length;
    }
    //파일 오퍼레이션 구조체
    //파일을 열 때 open()을 사용한다.
    static struct file_operations buzzer_fops = {
        .owner          = THIS_MODULE,
        .open           = buzzer_open,
        .write          = buzzer_write_byte,
        .release        = buzzer_release,
    };
    //모듈을 커널 내부로 삽입
    int buzzer_init(void)
    {
        int result;
        //문자 디바이스 드라이버를 등록한다.
        result = register_chrdev(BUZZER_MAJOR, BUZZER_NAME, &buzzer_fops);
        if(result < 0) { //등록실패

```

```

        printk(KERN_WARNING"Can't get any major\n");
        return result;
    }
    //major 번호를 출력한다
    printk(KERN_WARNING"Init module, Buzzer Major
number : %d\n",BUZZER_MAJOR);
    return 0;
}
//모듈을 커널에서 제거
void buzzer_exit(void)
{
    //문자 디바이스 드라이버를 제거한다
    unregister_chrdev(BUZZER_MAJOR, BUZZER_NAME);
    printk(KERN_INFO"driver: %s DRIVER EXIT\n", BUZZER_NAME);
}
module_init(buzzer_init);
module_exit(buzzer_exit);
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_LICENSE("Dual BSD/GPL");

```

Makefile

```
CC      = /usr/local/arm/arm-2010q1/bin/arm-none-eabi-gcc
```

```
obj-m := buzzer.o
```

```
KDIR := /Android/linux-2.6.32-hanback
```

```
PWD := $(shell pwd)
```

default:

```
$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
rm -f default
```

clean:

```
rm -f *.ko
rm -f *.o
rm -f *.mod.*
rm -f *.cmd
```

(2) 디바이스 드라이버 (7-segment) (/Android/drivers/242.seg_driver/)

segment.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <asm/fcntl.h>
#include <linux/ioport.h>
#include <linux/delay.h>
#include <asm/ioctl.h>
#include <asm/uaccess.h>
#include <asm/io.h>

#define DRIVER_AUTHOR "hanback"
#define DRIVER_DESC "7-Segment program"
#define SEGMENT_MAJOR 242
#define SEGMENT_NAME "SEGMENT"
#define SEGMENT_MODULE_VERSION "SEGMENT PORT V0.1"
#define SEGMENT_ADDRESS_GRID 0x88000030
#define SEGMENT_ADDRESS_DATA 0x88000032
#define SEGMENT_ADDRESS_1 0x88000034
#define SEGMENT_ADDRESS_RANGE 0x1000
#define MODE_0_TIMER_FORM 0x0
#define MODE_1_CLOCK_FORM 0x1

static unsigned int segment_usage = 0;
static unsigned long *segment_data;
static unsigned long *segment_grid;
static int mode_select = 0x0;
static int index = 0;
static int index2 = 0;

int
segment_open(struct inode *inode, struct file *filp)
{
    if( segment_usage != 0 ) return -EBUSY;
    //가상주소 맵핑
```

```

        segment_grid      = ioremap(SEGMENT_ADDRESS_GRID,
SEGMENT_ADDRESS_RANGE);
        segment_data      = ioremap(SEGMENT_ADDRESS_DATA,
SEGMENT_ADDRESS_RANGE);
        if(!check_mem_region((unsigned long)segment_data, SEGMENT_ADDRESS_RANGE)
            && !check_mem_region((unsigned long)segment_grid,
SEGMENT_ADDRESS_RANGE))
        {
            request_region((unsigned long)segment_grid, SEGMENT_ADDRESS_RANGE,
SEGMENT_NAME);
            request_region((unsigned long)segment_data, SEGMENT_ADDRESS_RANGE,
SEGMENT_NAME);
        }
        else      printk("driver: unable to register this!!!!\n");
        segment_usage = 1;
        return 0;
}

int
segment_release(struct inode *inode, struct file *filp)
{
    iounmap(segment_grid);
    iounmap(segment_data);
    release_region((unsigned long)segment_data, SEGMENT_ADDRESS_RANGE);
    release_region((unsigned long)segment_grid, SEGMENT_ADDRESS_RANGE);
    segment_usage = 0;
    return 0;
}

unsigned short
Getsegmentcode(short x)
{
    unsigned short code;
    switch(x)
    {
        //case 0x0~0x9 : 일반 숫자 출력용
        case 0x0 : code = 0xfc;          break;
        case 0x1 : code = 0x60;          break;
        case 0x2 : code = 0xda;          break;
        case 0x3 : code = 0xf2;          break;
        case 0x4 : code = 0x66;          break;
        case 0x5 : code = 0xb6;          break;
    }
}

```



```

        case 0x6 : code = 0xbe;          break;
        case 0x7 : code = 0xe4;          break;
        case 0x8 : code = 0xfe;          break;
        case 0x9 : code = 0xf6;          break;
        //case -1~-6 : 임의로 만든 segment 모양 출력용
        case -1 : code = 0x80;            break;
        case -2 : code = 0x0C;            break;
        case -3 : code = 0x10;            break;
        case -4 : code = 0x60;            break;
        case -5 : code = 0x00;            break;
        case -6 : code = 0x02;            break;
        default : code = 0;               break;
    }
    return code;
}

ssize_t
segment_write(struct file *inode, const char *gdata, size_t length, loff_t *off_what)
{
    unsigned char    data[6];
    unsigned char    digit[6] = {0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
    int              i, num, ret;
    int              count = 0, temp1, temp2;
    int              temp3=0;
    int              temp4=0;
    int              temp5=0;
    ret              = copy_from_user(&num, gdata, 4);
    count            = num;
    printk("!!! if num < 0 count is %d\n", count);
    //음수일때, temp3 는 count 에 -1 을 곱해 나머지 연산에서 문제가 없도록 만들고,
    //Getsegmentcode 함수 호출시, -1 을 또 의도적으로 곱해 음수로 만든 후, 미리
지정해놓은
    //음수일때의 Segment 모양을 불러온다.
    if(num<0)
    {
        temp3 = count * (-1);
        data[5] = Getsegmentcode(temp3 / 100000 * -1);
        temp1   = temp3 % 100000;
        data[4] = Getsegmentcode(temp1 / 10000 * -1);
        temp2   = temp1 % 10000;
    }
}

```

```

data[3] = Getsegmentcode(temp2 / 1000* -1);
temp1  = temp2 % 1000;
data[2] = Getsegmentcode(temp1 / 100* -1);
temp2  = temp1 % 100;
data[1] = Getsegmentcode(temp2 / 10 * -1);
data[0] = Getsegmentcode(temp2 % 10 * -1);
    switch (mode_select)
    {
        case MODE_0_TIMER_FORM:          break;
        case MODE_1_CLOCK_FORM:
            data[4] += 1;
            data[2] += 1;
            break;
    }
    for(i=0; i<6; i++)
    {
        *segment_grid  = digit[i];
        *segment_data  = data[i];
        mdelay(1);
    }
}

//100 만보다 큰 경우(999999 를 벗어난 경우), segment 에서 사용하지 않는다는 점을
활용해서
//특수하게 출력해야 하는경우를 100 만 이상일때로 정함. 숫자가 들어오면, 100 만의
나머지연산을 이용해 원래 출력하고자 하는 수를
//가져온다. 한자리수일때, 2 자리수일때를 나누어서 출력한다.
//이 부분은 정답일 때, 한칸씩 움직이는 부분의 정답이다.
else if(num > 1000000 && num < 2000000)
{
    for(i=0; i < 6; i++) data[i] = 0x00;
    temp3 = count;
    temp3 = temp3 % 1000000;
    if(temp3 < 10){
        switch(index){
            case 0 : data[0] = Getsegmentcode(temp3);
index2++;          break;
            case 1 : data[1] = Getsegmentcode(temp3);
index2++;          break;

```

```

                                case 2 :      data[2] = Getsegmentcode(temp3);
index2++;      break;
                                case 3 :      data[3] = Getsegmentcode(temp3);
index2++;      break;
                                case 4 :      data[4] = Getsegmentcode(temp3);
index2++;      break;
                                case 5 :      data[5] = Getsegmentcode(temp3);
index2++;      break;
                                }
                                if(index2 % 200 == 0 && index2 != 0)      index++;
                                if(index2 == 1200){index = 0; index2 = 0;}
                                }
                                //2 자리 수의 정답이 출력되는 부분
                                else
                                {
                                        temp4 = temp3 / 10;
                                        temp3 = temp3 % 10;
                                        switch(index){
                                                case 0 :      data[0] = Getsegmentcode(temp3); data[1]
= Getsegmentcode(temp4);      index2++;      break;
                                                case 1 :      data[1] = Getsegmentcode(temp3); data[2]
= Getsegmentcode(temp4);      index2++;      break;
                                                case 2 :      data[2] = Getsegmentcode(temp3); data[3]
= Getsegmentcode(temp4);      index2++;      break;
                                                case 3 :      data[3] = Getsegmentcode(temp3); data[4]
= Getsegmentcode(temp4);      index2++;      break;
                                                case 4 :      data[4] = Getsegmentcode(temp3); data[5]
= Getsegmentcode(temp4);      index2++;      break;
                                                case 5 :      data[5] = Getsegmentcode(temp3);
index2++;      break;
                                        }
                                        if(index2 % 200 == 0 && index2 != 0)      index++;
                                        if(index2 == 1200){index = 0; index2 = 0;}
                                }
                                switch (mode_select)
                                {
                                        case MODE_0_TIMER_FORM:      break;
                                        case MODE_1_CLOCK_FORM:
data[4] += 1;

```

```

        data[2] += 1;
        break;
    }
    for(i=0; i<6; i++)
    {
        *segment_grid = digit[i];
        *segment_data = data[i];
        mdelay(1);
    }
}
//틀렸을 때, 출력되는 부분이다. SegmentControl(2xx00xx, x=1~99)
//사용자 입력값 - 결과값
else if(num > 2000000 && num < 3000000)
{
    for(i=0; i<6; i++) data[i]=0x00; //초기화
    temp3 = count;
    temp3 = temp3 % 2000000; //이 연산으로 xx00xx 와 같은 숫자가
    된다.
    temp1 = temp3 / 10000; //이 연산으로 xx00xx 중 앞
    xx(입력값)을 temp1 에 저장한다.
    temp2 = temp3 % 10000; //이 연산으로 xx00xx 중 뒤
    xx(정답)을 temp2 에 저장한다.
    if(temp1 / 10 == 0) temp4 = 0x00;
    else temp4 = Getsegmentcode(temp1 / 10);
    //만약 temp1(정답)이 1 의 자리일때, 10 의 자리 부분에 0x00 코드(아무것도
    출력안함)을 입력한다.
    if(temp2 / 10 == 0) temp5 = 0x00;
    else temp5 = Getsegmentcode(temp2 / 10);
    //만약 temp2(사용자입력값)이 1 의 자리일때, 10 의 자리 부분에
    0x00 코드를 입력한다.
    switch (index)
    {
    case 0 : data[0] = Getsegmentcode(temp1 % 10);
            data[1] = temp4; data[2] = Getsegmentcode(-
6);
            data[3] = Getsegmentcode(temp2 % 10);
            data[4] = temp5;
            index2++; break;
    case 1 : data[1] = Getsegmentcode(temp1 % 10);

```

```

        data[2] = temp4; data[3] = Getsegmentcode(-6);
        data[4] = Getsegmentcode(temp2 % 10);
        data[5] = temp5;
        index2++; break;
    case 2 :    data[2] = Getsegmentcode(temp1 % 10);
                data[3] = temp4; data[4] = Getsegmentcode(-
6);

                data[5] = Getsegmentcode(temp2 % 10);
                index2++; break;
    case 3 :    data[3] = Getsegmentcode(temp1 % 10);
                data[4] = temp4; data[5] = Getsegmentcode(-
6);

                index2++; break;
    case 4 :    data[4] = Getsegmentcode(temp1 % 10);
                data[5] = temp4;
                index2++; break;
    case 5 :    data[5] = Getsegmentcode(temp1 % 10);
                index2++; break;
    }
    if(index2 % 200 == 0 && index2 != 0)    index++;
    if(index2 == 1200){index = 0; index2 = 0;}
    switch (mode_select)
    {
        case MODE_0_TIMER_FORM:            break;
        case MODE_1_CLOCK_FORM:
            data[4] += 1;
            data[2] += 1;
            break;
    }
    for(i=0; i<6; i++)
    {
        *segment_grid    = digit[i];
        *segment_data    = data[i];
        mdelay(1);
    }
}
//index 초기화용
else if(num == 3000000)
{

```

```

        index = 0; index2 = 0;
    }
    //일반 숫자가 출력되는 부분이다. SegmentControl(1~999999)
    else if(num > 0 && num < 1000000)
    {
        data[5] = Getsegmentcode(count / 100000);
        temp1   = count % 100000;
        data[4] = Getsegmentcode(temp1 / 10000);
        temp2   = temp1 % 10000;
        data[3] = Getsegmentcode(temp2 / 1000);
        temp1   = temp2 % 1000;
        data[2] = Getsegmentcode(temp1 / 100);
        temp2   = temp1 % 100;
        data[1] = Getsegmentcode(temp2 / 10);
        data[0] = Getsegmentcode(temp2 % 10);
        switch (mode_select)
        {
            case MODE_0_TIMER_FORM:           break;
            case MODE_1_CLOCK_FORM:
                data[4] += 1;
                data[2] += 1;
                break;
        }
        for(i=0; i<6; i++)
        {
            *segment_grid    = digit[i];
            *segment_data    = data[i];
            mdelay(1);
        }
    }
    *segment_grid = ~digit[0];
    *segment_data = 0;
    return length;
}

static int
segment_ioctl(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
    switch(cmd)
    {

```

```

        case MODE_0_TIMER_FORM:
            mode_select = 0x00;
            break;
        case MODE_1_CLOCK_FORM:
            mode_select = 0x01;
            break;
        default:
            return -EINVAL;
    }
    return 0;
}

struct file_operations segment_fops =
{
    .owner          = THIS_MODULE,
    .open           = segment_open,
    .write          = segment_write,
    .release        = segment_release,
    .ioctl          = segment_ioctl,
};

int
segment_init(void)
{
    int result;
    result = register_chrdev(SEGMENT_MAJOR, SEGMENT_NAME, &segment_fops);
    if (result < 0)
    {
        printk(KERN_WARNING "Can't get any major\n");
        return result;
    }
    printk(KERN_INFO "Init Module, 7-Segment Major Number : %d\n",
SEGMENT_MAJOR);
    return 0;
}

void
segment_exit(void)
{
    unregister_chrdev(SEGMENT_MAJOR, SEGMENT_NAME);
    printk("driver: %s DRIVER EXIT\n", SEGMENT_NAME);
}

```

```
module_init(segment_init);
module_exit(segment_exit);
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_LICENSE("Dual BSD/GPL");
```

Makefile

```
CC      = /usr/local/arm/arm-2010q1/bin/arm-none-eabi-gcc
obj-m   := segment.o
KDIR    := /Android/linux-2.6.32-hanback
PWD     := $(shell pwd)
default :
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
    rm -f default
clean   :
    rm -f *.ko
    rm -f *.o
    rm -f *.mod.*
    rm -f *.cmd
```

(3) JNI(ESTermProject)

(/Android/android-ndk-r5b/apps/ESTermProject/project/jni/)

ArrayAdder.c (ESTermProject.so 로 컴파일된다.)

```
#include <string.h>
#include <jni.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <sys/mman.h>
#include <errno.h>
#include <android/log.h>
#include <time.h>

//jintArray sumArray =NULL;
```



```
//buzzer Control
```

```
jint
```

```
Java_ac_kr_kgu_esproject_ArrayAdderActivity_BuzzerControl(JNIEnv* env, jobject thiz, jint value)
```

```
{  
    int fd, ret;  
    int data = value;  
    fd = open("/dev/buzzer", O_WRONLY);  
    if(fd < 0) return -errno;  
    ret = write(fd, &data, 1);  
    close(fd);  
    if(ret == 1) return 0;  
    return -1;  
}
```

```
//SegmentControl
```

```
jint
```

```
Java_ac_kr_kgu_esproject_ArrayAdderActivity_SegmentControl(JNIEnv* env, jobject thiz, jint data)
```

```
{  
    int dev, ret;  
    dev = open("dev/segment", O_RDWR | O_SYNC);  
    if(dev != -1)  
    {  
        ret = write(dev, &data, 4);  
        close(dev);  
    }  
    else  
    {  
        __android_log_print(ANDROID_LOG_ERROR, "SegmentActivity", "Device  
Open ERROR!Wn");  
        exit(1);  
    }  
    return 0;  
}
```

```
//SegmentIOControl
```

```
jint
```

```
Java_ac_kr_kgu_esproject_ArrayAdderActivity_SegmentIOControl(JNIEnv* env, jobject thiz, jint data)
```

```
{
```

```

int      dev, ret;
dev = open("/dev/segment", O_RDWR | O_SYNC);
if(dev != -1)
{
    ret = ioctl(dev, data, NULL, NULL);
    close(dev);
}
else
{
    __android_log_print(ANDROID_LOG_ERROR, "SegmentActivity", "Device
Open ERROR!\n");
    exit(1);
}
return 0;
}

//CreateArray 입력받은 배열 value 값만큼의 길이의 배열을 만들어서
//랜덤으로 숫자를 넣고, jintArray sumArray 를 반환한다.
jintArray
Java_ac_kr_kgu_esproject_ArrayAdderActivity_CreateArray(JNIEnv* env, jobject thiz, jint value)
{
    int i;
    jintArray sumArray = (*env)->NewIntArray(env, value);           //value 만큼
    길이의 배열을 생성한다.
    jint * array = (*env)->GetIntArrayElements(env, sumArray, 0);   //sumArray 의
    배열요소들을 변경하기 위한 작업
    //직접적으로 sumArray 를 건드리는게 아닌, array 를 이용해서 sumArray 의
    배열요소들을 변경한다.
    srand(time(NULL));
    for(i = 0; i < value; i++)
    {
        array[i] = (rand() % 9) + 1; //1~9 까지의 숫자를 출력한다.
    }
    (*env)->ReleaseIntArrayElements(env, sumArray, array, 0);
    return sumArray;
}

//입력받은 배열 sumArray 와 사용자가 입력한 결과값 value 를 입력받아 합을 비교한 뒤,
//결과값을 return 하는 함수, 정답은 1 을 리턴하고 오답일시 0 을 리턴한다.
jint

```

```

Java_ac_kr_kgu_esproject_ArrayAdderActivity_CompareArray(JNIEnv* env, jobject thiz, jint
value, jintArray sumArray)
{
    int sum = 0;
    int i;
    jsize len = (*env)->GetArrayLength(env, sumArray);           //배열의 길이를
저장하는 len 변수
    jint * array = (*env)->GetIntArrayElements(env, sumArray, 0); //sumArray 의
요소들을 받아오기 위한 작업
    for(i = 0; i < len; i++)
    {
        //sumArray 의 길이만큼 요소들을 받아와 sum 에 저장한다.
        sum += array[i];
    }
    (*env)->ReleaseIntArrayElements(env, sumArray, array, 0);
    if( value == sum )
        return 1;
    return 0;
}

```

Android.mk

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := ESTermProject
LOCAL_SRC_FILES := ArrayAdder.c
LOCAL_LDLIBS += -llog

```

```

include $(BUILD_SHARED_LIBRARY)

```

Application.mk (/Android/android-ndk-r5b/apps/ESTermProject/)

```

APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES := ESTermProject

```

(4) ArrayAdderActivity

ArrayAdderActivity.java

```

package ac.kr.kgu.esproject;
import android.app.Activity;
import android.os.Bundle;

```

```

import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Timer;
import java.util.TimerTask;
public class ArrayAdderActivity extends Activity {
    static {
        System.loadLibrary("ESTermProject");
    };
    static TimerTask tt;
    /**
     * 7-segment 부분
     */
    BackThread thread = new BackThread();
    int flag = -1;
    boolean stop = false;
    int count = 0;
    /**
     * JNI 함수들
     */
    public native int BuzzerControl(int value);
    public native int SegmentControl(int data);
    public native int SegmentIOContrl(int data);
    public native int[] CreateArray(int value);
    public native int CompareArray(int value, int[] sumArray);
    private ArrayAdapter<CharSequence> adapter;
    private ArrayAdapter<String> adapter2;
    private Spinner spinner;
    public int[] array;
    public int[] sumArray; // 덧셈 비교를 위한 랜덤 수를 저장하는 배열
    int BuzData; // 부저의 울림 여부를 정하기 위한 변수
    int lengthResult;
    String line = null;
    int index = 0; // 반복문 실행시 쓰기위함. segment 에서 사용

```

```

int sum;           // 사용자가 입력한 덧셈결과값
int realsum=0;     //실제 배열 합의 결과로 나온 진짜 덧셈 결과
int wrongsum=0;    //덧셈 결과가 틀렸을 때 들어가는 값
int result = 0;    // 배열 총 합과 사용자가 입력한 값의 참 거짓 여부 비교 변수
boolean exitOuterLoop = false;    //중첩 루프문을 빠져나오기 위해 쓰는 boolean
타입의 변수, 항상 쓰고 난 후 false 로 초기화

```

@Override

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    BuzData = 0;
    // 배열의 크기를 생성하는 spinner 생성
    spinner = (Spinner) findViewById(R.id.arraySpinner);
    adapter = ArrayAdapter.createFromResource(this, R.array.indexAmount,
        android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
    // 레이아웃들의 가시성 관리 변수
    final View vision1_1 = (View) findViewById(R.id.vision1_1);
    final View vision1_2 = (View) findViewById(R.id.vision1_2);
    final View vision2 = (View) findViewById(R.id.vision2);
    // 배열의 랜덤값의 결과를 화면에 출력해주는 TextView
    final TextView arrayResult = (TextView) findViewById(R.id.arrayResult);
    final LinearLayout vision1 = (LinearLayout) findViewById(R.id.vision1);
    final ArrayList<String> list = new ArrayList<String>();
    final LinearLayout.LayoutParams param = (LinearLayout.LayoutParams)

```

vision1

```

        .getLayoutParams();
    // 결과값을 비교한 후 정답여부 출력
    final TextView yesOrNo = (TextView) findViewById(R.id.yesOrNo);
    // 사용자가 정답을 입력하는 EditText
    final EditText inputNum = (EditText) findViewById(R.id.inputNum);
    // 부저컨트롤
    BuzzerControl(BuzData);
    // 배열을 생성하는 버튼
    Button arrayCreate = (Button) findViewById(R.id.createButton);
    // Segment 스레드 시작
    thread.setDaemon(true);

```

```

thread.start();
arrayCreate.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        // arraylist 와 Textview 초기화
        list.clear();
        arrayResult.setText("");
        int number =
Integer.parseInt(spinner.getSelectedItem()
                    .toString());
        sumArray = CreateArray(number);
        Toast.makeText(getApplicationContext(), "Right here",
                    Toast.LENGTH_LONG).show();
        lengthResult = 0;
        array = new int[number];
        for (int i = 0; i < array.length; i++) {
            lengthResult += 45;
            list.add("배열 요소 #" + (i + 1) + " : " +
sumArray[i] + "₩n");
        }
        // 배열에 어떤 수가 저장되었는지 화면에
        출력하기 위해 레이아웃의 높이를 변경

        param.height = lengthResult;
        vision1.setLayoutParams(param);
    }
    // ArrayList 에 저장된 모든 요소 출력
    Iterator<String> iter = list.iterator();
    while (iter.hasNext()) {
        arrayResult.append(iter.next());
    }
    vision1.setVisibility(View.VISIBLE);
    vision1_1.setVisibility(View.VISIBLE);
    vision1_2.setVisibility(View.VISIBLE);
}

});
tt = timerTaskMaker();
final Timer timer = new Timer();
// ok 를 눌렀을 경우 정답을 알려주는 TextView 출력
Button ok = (Button) findViewById(R.id.okButton);
ok.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {

```

```

try{
    sum = Integer.parseInt(inputNum.getText().toString());
    result = CompareArray(sum, sumArray);
} catch (Exception e){
    result = -1;
}
if(sum > 100)    result = 2;

```

//result: 1=정답, 0=오답, -1=아무것도 입력이 안된 오류,

2=99 가 넘는 숫자가 들어왔을 경우

```

if (result == 1) {
    flag = 1;
    yesOrNo.setText("정확합니다.");
    showDialog(1);
    tt = timerTaskMaker();
    timer.schedule(tt, 0, 1000);
} else if (result == -1) {
    yesOrNo.setText("아무것도 입력이

```

안되었습니다.");

```

    BuzData = 0;
} else if (result == 2)    {
    yesOrNo.setText("정답은 1~99 사이의

```

숫자입니다.");

```

    BuzData = 0;
} else {
    realsum = 0;
    for(int i=0; i<sumArray.length;i++) realsum

```

+= sumArray[i];

```

    BuzData = 1;
    flag = 0;
    yesOrNo.setText("틀렸습니다.");
}
BuzzerControl(BuzData);
vision2.setVisibility(View.VISIBLE);

```

```

}

```

```

});

```

// clear 를 눌렀을 경우 첫 줄을 제외한 모든 부분이 숨겨짐, 배열도 사라짐

```

Button clear = (Button) findViewById(R.id.clearButton);
clear.setOnClickListener(new Button.OnClickListener() {

```

```

        public void onClick(View v) {
            tt.cancel();
            BuzData = 0;
            BuzzerControl(BuzData);
            vision1.setVisibility(View.GONE);
            vision1_1.setVisibility(View.GONE);
            vision1_2.setVisibility(View.GONE);
            vision2.setVisibility(View.GONE);
            lengthResult = 0;
            flag = -1;                //segment 대기상태로

            realsum = 0;              //전에 더했었던 실제
            배열의 합을 다시 0으로 초기화함
        }
    });
}

//부저를 일정 반복하며 울리게 하기 위해 쓰인 TimerTask, 매번 쓸때마다 재생성 하는
이유는
//한번 중지하면 다시 켤 때 되살릴 수 없기 때문에 Clear 후 다시 정답을 입력했을 때,
재생성하여 사용한다.
//계속해서 BuzData 가 1 일때는 0으로, 0 일때는 1로 바꾸면서 소리를 낸다.
    public TimerTask timerTaskMaker() {
        TimerTask tempTask = new TimerTask() {
            public void run() {
                if (BuzData == 1)
                    BuzData = 0;
                else
                    BuzData = 1;
                BuzzerControl(BuzData);
            }
        };
        return tempTask;
    }

    // 일정하게 출력되게끔 의도적으로 반복하는 메소드, 200 번 반복하여 사람 눈에는
    계속 불이 들어와있는 것처럼 보인다.
    public void repeatSegmentControl(int value) {
        for (int i = 0; i < 200; i++)
            SegmentControl(value);
        return;
    }

```



```

    }
    //결과값 표시의 대기상태 출력용, 대기상태때 썼던 segmentcode 를 재활용하여 마치
사람눈에는
    //네모난 직사각형 박스가 계속해서 들어와있는것처럼 보인다.
    public void repeatSegmentControl_Box() {
        for (int i = 0; i < 50; i++) {
            SegmentControl(-211114);
            SegmentControl(-233334);
            SegmentControl(-111111);
            SegmentControl(-333333);
        }
    }
    // segment 출력을 담당하는 스레드
    class BackThread extends Thread {
        // flag : -1 -> 대기상태, 1 -> 답이 맞았을때, 0 -> 틀렸을때
        public void run() {
            while (true) {
                SegmentIOContrl(0); // 시간은 쓸일이 없으므로
0 으로 고정한다.

                while (flag == 1) { // 답이 맞았을 때
                    SegmentControl(3000000);
                    try {
                        for(int i=0; i<3; i++){

                            repeatSegmentControl_Box();

                                thread.sleep(500);
                                if(flag != 1){
                                    exitOuterLoop =
true;

                                    break;
                                }
                            }
                        }
                    }
                    if(exitOuterLoop){
                        exitOuterLoop = false;
                        break;
                    }
                }
            }
            //위 for 문에서 3 번 반복한다. 만약
갑자기 clear 버튼이 들어오면, 완전히 루프문을 빠져나오기 위해 true 로

```

반복문을 완전히 정지시킨다.

```
//바꾼 후, 아래서 false 로 초기화하고
```

눌러지면 즉각적으로 멈춘다.

```
//간단히 말해서, Clear 버튼이
```

수 있도록 exitOuterLoop 를 끼워넣고, 정답 숫자를 출력한다.

```
//위와 같은 원리로 즉각적으로 멈출
```

```
for(int i=0; i<6; i++)  
{
```

```
repeatSegmentControl(sum+1000000);
```

```
//sum+1000000 을
```

함으로써 드라이버는 합을 출력하는 상태로 받아들여 설정한대로 출력한다.(한칸씩 이동)

```
thread.sleep(500);
```

```
if(flag != 1)
```

```
{
```

```
exitOuterLoop =
```

```
true;
```

```
break;
```

```
}
```

```
}
```

```
if(exitOuterLoop)
```

```
{
```

```
exitOuterLoop = false;
```

```
break;
```

```
}
```

```
} catch (Exception e) {
```

```
System.out.println("flag = 1 error");
```

```
}
```

```
}
```

```
//틀렸을때의 동작, 2 xx 00 xx 와 같은 숫자로 들어간다.
```

```
//앞의 xx 는 사용자가 입력한 값이고, 뒤 xx 는 실제
```

정답값이다.

```
while (flag == 0) {
```

```
SegmentControl(3000000);
```

```
try{
```

```
for(int i=0; i<3; i++){
```

```

repeatSegmentControl_Box();

thread.sleep(500);
if(flag != 0){
    exitOuterLoop =

break;
}
}

if(exitOuterLoop){
    exitOuterLoop = false;
    break;
}
wrongsum = realsum * 10000 +

sum + 2000000;

for(int i=0; i<6; i++){

repeatSegmentControl(wrongsum);

thread.sleep(500);
if(flag != 0){
    exitOuterLoop =

break;
}
}

if(exitOuterLoop){
    exitOuterLoop = false;
    break;
}
}
catch(Exception e)
{
    System.out.println("flag 0 Error");
}
}
while (flag == -1) {
    SegmentControl(3000000);

```

반복하면서 출력해야 하므로, switch 문으로
넘어가는 방식으로 만들었다.

//모양이 계속 바뀌는 대기를 계속해서

//index 가 하나씩 증가하면서 다음으로

```
switch (index) {
case 0:
    repeatSegmentControl((-555551));    index++; break;
case 1:
    repeatSegmentControl((-555515));    index++; break;
case 2:
    repeatSegmentControl((-555155));    index++; break;
case 3:
    repeatSegmentControl((-551555));    index++; break;
case 4:
    repeatSegmentControl((-515555));    index++; break;
case 5:
    repeatSegmentControl((-155555));    index++; break;
case 6:
    repeatSegmentControl((-255555));    index++; break;
case 7:
    repeatSegmentControl((-355555));    index++; break;
case 8:
    repeatSegmentControl((-535555));    index++; break;
case 9:
    repeatSegmentControl((-553555));    index++; break;
case 10:
    repeatSegmentControl((-555355));    index++; break;
case 11:
    repeatSegmentControl((-555535));    index++; break;
case 12:
    repeatSegmentControl((-555553));    index++; break;
case 13:
    repeatSegmentControl((-555554));    index = 0;
}
}
}
}
}
```

(5) Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="70dp"
            android:orientation="horizontal">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="배열 요소 개수"
                android:textSize="30sp"
                android:layout_marginTop="1dp"
                android:layout_marginRight="30dp"
                android:layout_marginLeft="5dp"
                android:layout_weight="1"/>
            <Spinner
                android:id="@+id/arraySpinner"
                android:textSize="25sp"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:entries="@array/indexAmount"
                android:layout_weight="1"/>
            <Button
                android:id="@+id/createButton"
                android:text="배열 생성"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"/>
        </LinearLayout>
    <LinearLayout
```

```

        android:id="@+id/vision1"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:orientation="horizontal"
        android:visibility="gone">
        <TextView
            android:id="@+id/arrayResult"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="5dp"
            android:textSize="25sp"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/vision1_1"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:orientation="horizontal"
        android:visibility="gone">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="덧셈 결과 : "
            android:textSize="30sp"
            android:layout_weight="1"
            android:layout_marginTop="8dp"
            android:layout_marginLeft="5dp"/>
        <EditText
            android:id="@+id/inputNum"
            android:layout_width="230dp"
            android:layout_height="wrap_content"
            android:hint="What is a result???"
            android:textSize="20sp"
            android:layout_marginLeft="80dp"
            android:layout_weight="1"
            android:inputType="number"
            android:maxLines="1"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/vision2"

```

```
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:orientation="horizontal"
        android:visibility="gone">
        <TextView
            android:id="@+id/yesOrNo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Exactly."
            android:textSize="30sp"
            android:layout_marginTop="8dp"
            android:layout_marginLeft="5dp"/>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/vision1_2"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:orientation="horizontal"
        android:visibility="gone">
        <Button
            android:id="@+id/okButton"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="확인"
            android:layout_weight="1"/>
        <Button
            android:id="@+id/clearButton"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Clear"
            android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
</RelativeLayout>
```

실행 결과

/* 실행 결과 화면 캡처 */