Mobile System Programing

project # : hardware interlocking system

모바일시스템공학과 김태경 32211203
모바일시스템공학과 유준혁 32212808

목차

2025.06.20

1. Project Overview (프로젝트 개요)

This project aims to control various devices (Keypad, Dipsw, Dotmatrix, LED, Piezo, 7-Segment, TextLCD) of Android 4.1 based embedded board (HBE-SM5-S4210-M3) through Android application. The device interface between the board and the FPGA was accessed through shell command and JNI.

1.1 Board Specifications & Development Environment Settings

1.1.1 Board Specifications

Category	Content
Board	HBE-SM5-S4210-M3
Device driver	FPGA
Kernel	Linux kernel 3.0.15
OS	Android 4.1 (“Jelly Bean”, API Level 16)

1.1.2 Development Environment Settings

Category	Content	Environment
Connect port	Base board - Power line Base board - Serial port Middle board - USB port (ADB)	Android board
Install kernel	Linux Kernel 3.0.15	
Install OS	Android 4.1 (“Jelly Bean”, API Level 16)	
Install development tool	arm-linux-gnueabi-gcc ADB & fastboot Android Studio	Assam server Windows
Check port	Device manager → Check com number → Set PuTTY (SSH) 115200, None	

1.3.2.1 How to make a kernel image

Category	Instruction	Environment
Install crosscomplier	cd linux export PATH=$PATH:/opt/arm-linux-gnueabi-gcc/bin	Assam server
Make menuconfig	ARCH=arm make menuconfig → Modify <> <M> <*>	
Make zImage	ARCH=arm make zImage	
Download locally	cd User/[name]/appdata/local/Android/sdk/platform tools scp [name]@assam.dankook.ac.kr:~/linux/arch/arm/boot/zImage .	Windows PowerShell

1.3.2.2 How to update kernel image

Category	Instruction	Environment
Update zImage	cd User/[name]/appdata/local/Android/sdk/platform tools .\adb devices .\fastboot.exe flash kernel zImage	Window PowerShell
Fastboot	fastboot reboot	Android board

2. Main concepts

2.1 Linux device driver concepts (리눅스 디바이스 드라이버의 개념)

In Linux, device drivers act as an interface between hardware and the kernel.

Because hardware cannot do anything by itself, the OS controls hardware resources and send and receives data through device drivers.

2.1.1 Device driver Type

Type	Content	Example
Character Device	Read and write data sequentially (1Bite)	Serial port, LED
Block Device	Process data in blocks	Hard disk, SSD, USB drive
Network Device	Send and receive network packets	Ethernet

2.1.1.1 Character Device

Function name	Content
open	Open and initialization device file
read	Read device data and send data to user
write	Get data from user and send data to device
release	Close device file and clean up

2.1.2 major & minor number and /dev node

2.1.2.1 major & minor number

Number	Content
major number	Identify the device driver
minor number	Separate the sub-device from the inside of the driver

2.1.2.2 /dev node

Devices are accessed in the form of a file system, such as /dev/device_name, in the user space.
To do this, the user must create a node (device file) in the /dev directory.

``` sudo mknod /dev/device_name c [major] [minor] ```

2.1.3 insmod, rmmod, dmesg

Category	Content	Command
insmod	Load compiled kernel module to kernel Run driver initialization function (module_init())	sudo insmod driver_name.ko
rmmod	Remove module in the kernel Run driver termination function (module_exit())	sudo rmmod mydriver
dmesg	Check the log output by the driver Good for debugging	dmesg | tail
```

2.2. Linux device driver (리눅스 디바이스 드라이버)

2.2.1 Keypad Device Driver (hanback_fpga_keypad.c)

Global variable	Content
keypad_ioremap	Keypad IO memory mapping
keypad_row_addr keypad_col_addr	low, column address pointer for detect button pressed
keypad_fpga_keycode[16]	key code mapping table
idev	input_dev structure for sending event to input subsystem

Function	Content
keypad_init()	Module initialization
keypad_open()	Keypad low, column pointer address initialization
mypollingfunction()	Check whether a key is entered at regular intervals
keypad_read()	Copy the last detected value into the user space
keypad_release()	Delete timer for release polling
keypad_exit()	iounmap() and release memory, remove input device

2.2.2 Dipsw Device Driver (fpga_dipsw.c)

Global variable	Content
dipsw_usage	Prevent duplicate opening
dipsw_ioremap	Pointer mapped to kernel address
dipaddr	Pointer for register access
value[4]	Array that stores read values

Function	Content
dipsw_init()	Register as misc device
dipsw_open()	Prevent duplicate opening
dipsw_read()	value[0] = ((*dipaddr & 0x00f0) >> 4) | ((*dipaddr & 0x000f) << 4); value[1] = ((*dipaddr & 0xf000) >> 8) >> 4 | ((*dipaddr & 0x0f00) >> 8) << 4;
dipsw_release()	iounmap() and release memory
dipsw_exit()	Unregister the device

2.2.3 Dotmatrix Device Driver (fpga_dotmatrix.c)

Global variable	Content
dot_usage	장치 중복 사용 방지 플래그
dot_ioremap	메모리 영역 커널 주소로 매핑한 결과
dot_row_addr	행 제어 레지스터 주소 (+0x40)
dot_col_addr	열 제어 레지스터 주소 (+0x42)

Function	Content
dot_open()	장치 파일 open 시 호출, 메모리 매핑, 초기화
dot_release()	장치 파일 close 시 호출, 메모리 해제, 메모리 반납
dot_write()	사용자로부터 받은 16진수 문자열 (20자)을 10개 숫자로 파싱하여 LED에 출력
dot_init()	모듈 초기화 함수
dot_exit()	모듈 종료 함수

2.2.4 LED Device Driver (fpga_led.c)

Global variable	Content
led_usage	중복 사용 방지 플래그
led_ioremap	ioremap()으로 매핑된 LED 장치 메모리 주소

Function	Content
fpga_led_open()	open 시 호출, 메모리 매핑, IO 영역 확보
fpga_led_release()	close 시 호출, 메모리 해제
fpga_led_write_byte()	사용자로부터 받은 1바이트 값을 LED 레지스터에 직접 씀
fpga_led_init()	모듈 초기화 시 /dev/fpga_led 생성
fpga_led_exit()	모듈 제거 시 장치 해제

2.2.5 Piezo Device Driver (fpga_piezo.c)

Global variable	Content
piezo_usage	드라이버 중복 open 방지 플래그
piezo_ioremap	piezo 장치의 메모리 매핑 주소
piezo_addr	실제 piezo 제어용 레지스터 주소

Function	Content
piezo_open()	open 시 호출, ioremap(), 메모리 매핑
piezo_release()	close 시 호출, iounmap(), 메모리 해제
piezo_write()	사용자로부터 1바이트 값을 받아 piezo에 직접 출력
piezo_init()	드라이버 초기화 시 호출
piezo_exit()	드라이버 종료 시 호출

2.2.6 7-Segment Device Driver (fpga_segment.c)

Global variable	Content
segment_usage	드라이버 중복 사용 방지 플래그
segment_grid	Grid 제어용 레지스터 (6자리 중 켜지는 자리를 결정)
segment_data	Segment에 표시할 숫자 값
segment_select	출력 모드 선택 변수

Function	Content
fpga_segment_open()	Open 시 호출, ioremap(), IO 메모리 요청
fpga_segment_release()	Close 시 호출 메모리 해제
Getsegmentcode()	0~F 까지의 숫자에 대응되는 7-Segment 코드값 반환
fpga_segment_write()	사용자로부터 6자리 정수를 받아 각 자릿수로 분해
fpga_segment_ioctl()	IOCTL 명령으로 mode_select 설정 (0: 일반, 1: 시계)
fpga_segment_init()	모듈 초기화, 장치 등록
fpga_segment_exit()	모듈 종료, 장치 해제

2.2.7 TextLCD Device Driver (fpga_textlcd.c)

Global variable	Content
textlcd_ioremap	LCD 제어용 메모리 매핑 주소
textlcd_usage	드라이버 중복 접근 방지 플래그

Function	Content
textlcd_open()	Open 시 호출, 메모리 매핑, 초기화 수행
textlcd_release()	Close 시 호출, iounmap(), 메모리 해제
textlcd_write()	문자열 전체를 받아 writebyte()로 문자 단위 출력
textlcd_ioctl()	LCD 제어 명령 전달 (cursor shift, clear, set position 등)

2.3. JNI (Java Native Interface)

JNI는 Java code에서 C, C++ 등의 Native language의 함수들을 호출할 수 있게 해주는 interface입니다. Java에서 제공하지 않는 low level hardware 기능 (hardware device 제어, 속도 최적화, legacy C library 연동) 등에 사용됩니다.

2.3.1 Java code ↔ Native C code 연동 방식

I. Java code에 native method를 선언합니다.
II. System.loadLibrary("native-lib")를 사용하여 C library를 로드합니다.
III. javac, javac -h, javah 등의 C header file을 생성합니다.
IV. C, C++ file에서 JNIEXPORT를 사용하여 함수를 정의합니다.
V. .so로 compile하고, Java code에서 사용합니다.
VI. jniLibs/ directory나 CMake, NDK build system을 사용하여 .so file이 android app에 포함됩니다.
VII. Android app이 실행되면 System.loadLibrary()를 사용하여 해당 .so file이 로드됩니다.

2.3.2 Android app ↔ Hardware 통신 흐름

I. Android app (UI by Java code)
↓ JNI native function 호출
II. C JNI function
↓ open(), write(), ioctl() 등의 system call
III. Linux device driver
↓ LED, Keypad 등의 실제 Hardware 제어
IV. FPGA hardware circuit
↓
V. Hardware

2.4 FPGA 입출력 제어

FPGA (Field Programmable Gate Array) 는 하드웨어 회로를 소프트웨어적으로 구성할 수 있는 IC (Integrated Circuit) 로, FPGA에 미리 설계된 회로를 리눅스에서 제어하는 것입니다.

2.4.1. How to control the FPGA

I. 리눅스의 디바이스 드라이버는 /dev/fpga_textlcd, /def/fpga_led, /dev/fpga_buzzer 등의 디바이스 파일을 관리합니다. 디바이스 파일은 FPGA와 직접적으로 연결된 장치로, 디바이스 드라이버를 통해 FPGA를 제어할 수 있습니다.

II. 구체적으로 디바이스 드라이버는 커널 안에서 다음을 수행합니다.

   A. open() → 디바이스 파일 접근을 허용합니다.

   B. write() → 디바이스 파일을 통해 사용자의 데이터를 FPGA 레지스터로 전달합니다.

2.4.2 How to send the value to FPGA ex) 0x1FF1000

I. 일반적으로 FPGA는 특정 메모리 주소에 mapping되어 있으며, 리눅스에서는 ioremap()을 통해 이 주소에 접근할 수 있습니다.

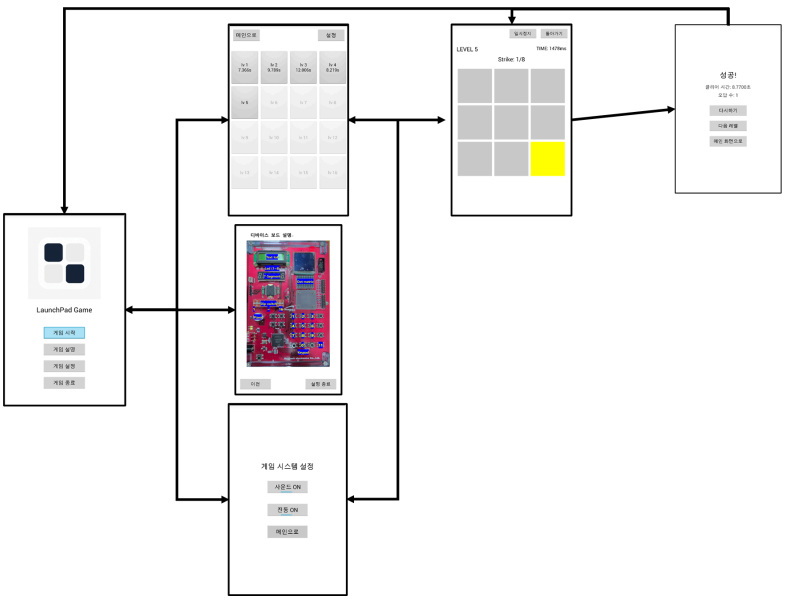
II. 다음으로 iowrite8(), iowrite32()를 통해 FPGA로 값을 전송하며, FPGA 내부 회로가 그 값을 해석해서 동작합니다.

2.4.3 Keypad, Dipsw, Dotmatrix, LED, Piezo, 7-Segment, TextLCD

Device	**MAC**	**Range**	**ioremap()**	**IO** 접근 함수 사용법법
TextLCD	0x05000000		textlcd_ioremap	*pointer = value (16bit)
Dotmatrix		0x1000	dot_ioremap	
Keypad			keypad_ioremap	*pointer = value (8bit)
Piezo	0x05000050		piezo_ioremap	
LED	0x05000020		led_ioremap	`*pointer = value
Dipsw	0x05000062		dipsw_ioremap	*pointer (16bit, read only)
7-Segment	0x05000030(Grid) 0x05000032(Data)		segment_grid segment_data	*pointer = value (16bit)

3. 시스템 구성

3.1 페이지 흐름 구조도



MainActivity
・Stage, Guide, Settings

StageActivity
・Main, Game

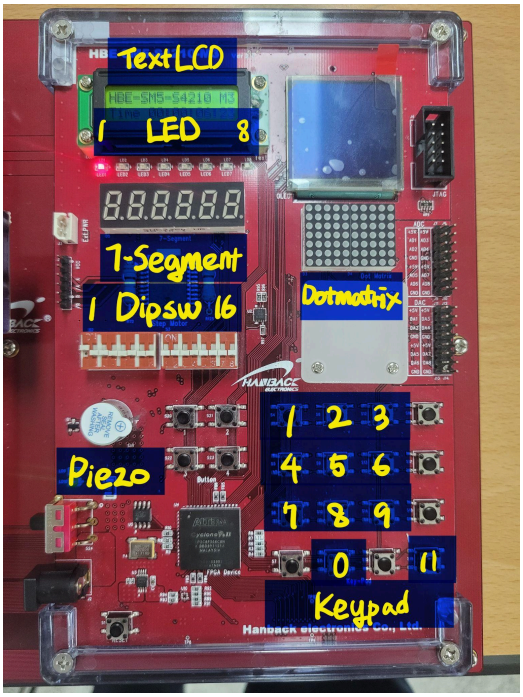
GameActivity
・Stage, Result

ResultActivity
・Game, Main

GuideActivity
・Main

SettingsActivity
・Main

3.2 하드웨어 구성도



TextLCD는 문자열을 입력할 경우 그대로 출력합니다.
・”Hello World”
→ Hello World

Dotmatrix는 왼쪽 아래에서 오른쪽 위로 비트 표기를 사용합니다.
・ 0x04, 0x00, 0x00, …
→ 왼쪽 위에서부터 4개 ON

Keypad는 실제로는 버튼이 다른 값을 반환하며 누를 때와 땔 때 2번 값을 반환합니다. 여기서는 수정하여 사용하였습니다.
・ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14
→ 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 11

Piezo는 파일에 값이 쓰이고 sleep하는 동안 음을 출력합니다.
LED는 1 ~ 8 까지 숫자의 입력에 따라 ON LED의 수가 결정됩니다.
・5
→ LED 5개 ON

Dipsw는 ON인 것 중 가장 우선순위가 높은 것을 출력해 Dotmatrix에 띄우도록 수정하였습니다.
・3, 5, 7, 10
→ 10 출력

7-Segment는 정적인 값을 짧게 띄우거나 타이머 기능 모두 사용할 수 있도록 수정하였습니다.

3.3 장치 사양 요약

Device	Control logic	Connection	Device file
TextLCD	FPGA	J5	/dev/fpga_textlcd
Dotmatrix		J6	/dev/fpga_dotmatrix
Keypad		J2	/dev/fpga_keypad
Piezo		J8, J9	/dev/fpga_piezo
LED		J4	/dev/fpga_led
Dipsw		J3	/dev/fpga_dipsw
7-Segment		J7	/dev/fpga_segment

---

4. 구현 과정

4.1. Android App 개발 요약

해당 프로젝트의 Android 애플리케이션은 사용자의 반사 신경을 시험하는 리듬 게임으로, 외부 하드웨어 장치와의 실시간 상호작용을 기반으로 설계되었습니다. Android API 16(Jelly Bean 4.1) 환경에서 개발되었으며, 앱은 총 다섯 개의 주요 Activity로 구성되어 각기 독립적인 기능과 하드웨어 제어 로직을 수행합니다.

특징으로, 터치스크린이 존재함에도 모든 조작은 3x3 키패드, 단일 키패드, DIP 스위치 등 물리적 입력 장치를 통해 이루어지도록 설계되었습니다. 사용자 입력뿐 아니라 출력도 LED, LCD, dot-matrix, buzzer, piezo 등을 통해 시각·청각·촉각으로 제공됩니다.

앱의 전체 흐름은 메인 화면(MainActivity)에서 시작하여, 스테이지 선택(StageActivity), 게임 진행(GameActivity), 결과 확인(ResultActivity) 순으로 이어지며, 설정(SettingsActivity)에서 전체 시스템 제어 옵션을 조정할 수 있습니다. 모든 하드웨어 동작은 JNI 및 디바이스 드라이버를 통해 /dev/fpga_* 인터페이스를 기반으로 구현됩니다.

4.1.1. MainActivity - 메인화면

앱 실행 시 첫 화면으로, 사용자는 키패드를 통해 원하는 기능 버튼을 선택하며, 물리 입력으로 모든 내비게이션을 수행합니다.

구성 요소:

- 앱 로고
- 타이틀 텍스트
- 버튼: [게임 시작] / [게임 설명] / [게임 설정] / [게임 종료]

조작 방식:

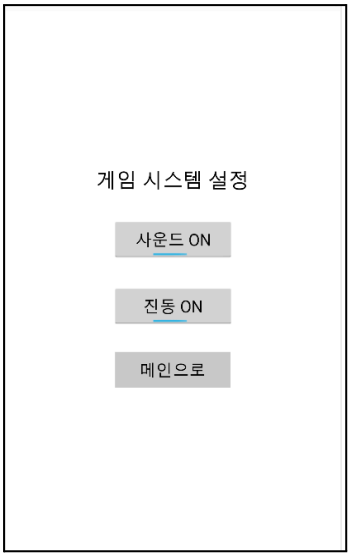
- 단일 키패드 1: 커서 이동
- 단일 키패드 2: 선택 버튼

하드웨어 연동

- 키패드 입력: /dev/keypad1, /dev/keypad2 를 통해 커서 이동 및 선택 처리

4.1.2. SettingsActivity - 설정 페이지

메인 화면에서 [게임 설정] 버튼을 누르면 나타나는 페이지입니다. 게임 진행 중 진동 및 사운드의 on/off 설정을 할 수 있습니다.



구성 요소

- 진동 설정 토글
- 사운드 설정 토글
- 버튼: [메인으로]

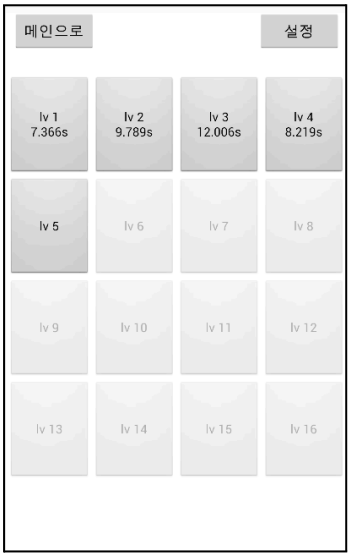
조작 방식

- 단일 키패드 1: 커서 이동
- 단일 키패드 2: 현재 항목 토글 또는 선택

하드웨어 연동

- Buzz: 진동 설정을 ON할 때 1초 진동
- Pizeo: 사운드 설정을 ON할 때 짧은 소리 출력
- 키패드 입력: 커서 조작 및 상태 반영
- 설정 상태: SharedPreferences로 저장 및 전체 앱에서 반영

4.1.3 StageActivity - 레벨(난이도) 선택 화면



메인화면에서 [게임 시작] 버튼을 누르면 나타나는 페이지로 1~16단계 중 하나를 선택하여 게임을 시작할 수 있습니다. 클리어된 레벨은 최고 기록의 시간이 표기되며, dip-switch를 통한 레벨 선택을 지원합니다.

구성 요소

- 상단 버튼: [메인으로] / [설정]
- 4X4 Grid 형태의 Lv 선택 버튼

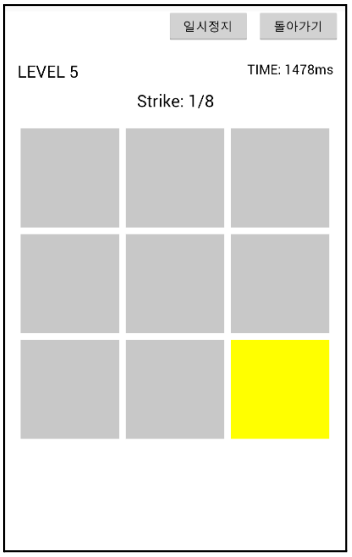
조작 방식:

- 단일 키패드 1 & 2: 커서 이동 / 현재 항목 선택
- Dip-Switch: 현재 DIP 상태에서 가장 낮은 비트를 레벨로 지정

하드웨어 연동

- dot-matrix: DIP 상태에 따라 Lv X 출력
- DIP 스위치: /dev/dip에서 읽은 값을 통해 레벨 인식
- LCD: 현재 커서 또는 DIP 선택 레벨을 출력
- 키패드: Lv 버튼 간 이동 및 실행

4.1.4.GameActivity - 게임 플레이 화면



레벨 선택 페이지에서 선택된 레벨을 기반으로 게임을 실행하는 페이지입니다. 각각 버튼을 통한 시퀀스 입력 → 타이머 제한 → 정답/오답 판별 후 최종적으로 성공/실패 여부를 결정합니다.

구성 요소

- 상단 버튼: [일시정지/이어하기] / [돌아가기]
- 상단 게임 정보: Lv 정보 / 경과 시간 / strike 수 표시
- 3X3 Grid 버튼

조작 방식

- 3X3 키패드: 게임 중 표시된 3X3 버튼 입력
- 단일 키패드 1&2: 각각 상단 버튼 조작

하드웨어 연동

- 7-segment: 게임 시작 전 카운트다운, 실시간 게임 플레이타임 표시
- LCD: 게임 시작 전 "Ready", "Go!", 게임 중 시퀀스에 따라"Good", "Wrong", "Miss" 텍스트 출력
- LED: Strike 수 점등 (1~8개), 실패 시 8개 모두 점멸
- Buzz: 오답/미입력 시 0.1초, 실패 시 1초간 진동
- Pizeo: 시작 알림음 (0.3초)

게임 흐름

1) 카운트다운(3초)
    - 7-segment: 3 → 2 → 1
    - LCD: “Ready” → “Go!”
2) 게임 시작
    - 버튼 점등 시퀀스 표시 (버튼이 노란색으로 표시 / 랜덤)
    - 제한 시간 내 입력 대기 (Lv별 0.75 ~ 0.3초)
3) 사용자 입력
    - 정답 시 (버튼이 파란색으로 표시): 다음 시퀀스 진행
    - 오답 시 (버튼이 빨간색으로 표시): 시퀀스 대기 / strike 수 증가
    - 미입력 시 (버튼이 빨간색으로 표시): 다음 시퀀스 진행 / strike 수 증가
4) 게임 종료 조건
    - 성공: 모든 시퀀스 완료
    - 실패: Strike 8회 도달
5) 결과 전달
    - ResultActivity로 이동 (성공 / 실패 및 타이머 기록 정보 포함)

4.1.5 ResultActivity - 결과 화면

게임 플레이가 후 게임 결과를 보여주는 페이지입니다. 게임 결과의 정보를 보여주며, 다음 동작을 버튼으로 유도합니다.

구성요소

- 결과 텍스트: “Clear” / “Fail”
- 경과 시간 표시: n.0000초
- Strike 횟수: 오답 수 : n
- 버튼: [다시하기] / [다음 레벨] / [메인으로]

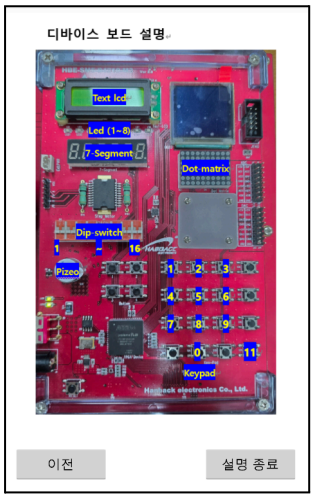
조작 방식

- 단일 키패드 1 & 2: 버튼 간 커서 이동 / 선택 실행

하드웨어 연동

- LCD: 전체 결과 정보 출력 (“Clear” / “Fail”)
- 7-segment: 성공 시 최종 기록 시간 표시 (n.0000초)
- Buzz: 성공 + 신기록 → 0.3초 × 3 진동 / 실패 시 1초 진동
- Pizeo: 성공/실패 음향 재생 (1~2초)

4.1.6 GuideActivity - 게임 설명 화면



메인화면에서 [게임 설명] 버튼을 누르면 옮겨지는 페이지로 게임 룰과 하드웨어 조작 등의 설명을 이미지 파일을 버튼으로 넘겨가며 진행하는 가이드 페이지입니다.

구성요소

- 이미지 파일(1 ~ 5장): 설명서 이미지 파일
- 버튼: [메인으로] / [이전] / [다음]

하드웨어 연동

- 단일 키패드 1: 버튼 간 커서 이동
- 단일 키패드 2: 선택 실행

종합 요약

해당 애플리케이션은 단순한 안드로이드 기반 리듬 게임을 넘어서, 모든 조작을 외부 하드웨어 입력 장치만으로 수행할 수 있도록 완전히 통합된 시스템으로 설계되었습니다. 터치스크린이 탑재된 환경임에도 불구하고, 실제 사용자 조작은 키패드, DIP 스위치 등 물리적 장치만을 통해 이루어지며, 이는 사용자 인터페이스 설계 측면에서 차별화된 목표를 반영했습니다.

4.2. 하드웨어 제어 로직 설계

JNI 구조는 각 디바이스별로 표준 Unix open()/read()/write()/close() 시스템 콜을 사용하며, Android Java → JNI C → /dev/fpga_* 디바이스 파일로 연결되는 형태로 설계되었습니다. 모든 장치는 직접 메모리 매핑 없이 파일 기반 입출력을 수행하며, LED/Segment/DotMatrix/LCD 등은 write() 중심, DIP/Keypad는 read() 중심으로 구성되어 있습니다.

4.2.1 LED (fpga_led)

목적	왼쪽부터 n개의 LED 점등
파일 경로	/dev/fpga_led
JNI 함수	Device.led(String level) → native void led(String)
C 함수	Java_com_example_test5_Device_led()
write 방식	write(fd, &val, 1) (8bit 값 전송)
데이터 형식	unsigned char (0x01 ~ 0xFF)
설명	입력값 n에 대해 왼쪽부터 n개의 비트를 1로 설정하여 점등
예시	Device.led("3") → val = 0x07 (0000 0111)

4.2.2 DIP Switch (fpga_dipsw)

목적	16bit DIP 스위치 상태 읽기
파일 경로	/dev/fpga_dipsw
JNI 함수	Device.dip() → native String dip()
C 함수	Java_com_example_test5_Device_dip()
read 방식	read(fd, &value, 4)
데이터 형식	unsigned short[2]
설명	2바이트 스위치 상태를 문자열 16자리로 변환 후 반환
예시	DIP 상태 0x8000 → "1000000000000000" 반환

4.2.3 7-Segment Display (fpga_segment)

목적	6자리 숫자 출력 또는 카운트 모드
파일 경로	/dev/fpga_segment
JNI 함수	Device.segment(String value, String mode)
C 함수	Java_com_example_test5_Device_segment()
write 방식	write(fd, &value, sizeof(int))
데이터 형식	int (최대 6자리 숫자)
설명	mode="1"일 경우 카운터 실행, mode="0"은 지정 숫자 출력
예시	segment("123456", "0") → "123456" 표시

4.2.4 Dot-Matrix (fpga_dotmatrix)

목적	숫자(1~16) 또는 전체 OFF 상태 출력
파일 경로	/dev/fpga_dotmatrix
JNI 함수	Device.dotWrite(String value)
C 함수	Java_com_example_test5_Device_dotWrite()
write 방식	write(fd, output, 20)
데이터 형식	char[20] (16진수 문자열로 구성된 dot 행렬)
설명	숫자별 dot 패턴을 HEX 문자열로 변환 후 출력
예시	dotWrite("3") → "00003E223E..." 전송

4.2.5 Text LCD (fpga_textlcd)

목적	문자열 출력 (최대 32자)
파일 경로	/dev/fpga_textlcd
JNI 함수	Device.lcd(String text)
C 함수	Java_com_example_test5_Device_lcd()
write 방식	write(fd, text, strlen(text))
데이터 형식	char* (ASCII 문자열)
설명	문자열을 그대로 출력
예시	lcd("Ready\nGo!") → Ready / Go! 표시

4.2.6 Piezo Buzzer (fpga_piezo)

목적	지정된 음계로 피에조 재생
파일 경로	/dev/fpga_piezo
JNI 함수	Device.piezo(String value)
C 함수	Java_com_example_test5_Device_piezo()
write 방식	write(fd, value, 1)
데이터 형식	char ("0", "1", "2")
설명	"0": 음 없음, "1": 성공음, "2": 실패음
예시	piezo("1") → 짧은 소리 재생

4.2.7 Keypad Input (input event)

목적	키패드 입력 감지
파일 경로	/dev/input/event4
JNI 함수	Device.getKeypadCode()
C 함수	Java_com_example_test5_Device_getKeypadCode()
read 방식	read(fd, &ev, sizeof(ev))
데이터 형식	struct input_event
설명	입력 발생 시 ev.code 값을 키 매핑하여 반환
예시	키패드 1번 → raw code 2 → mapped code 1

4.3. JNI 설계 및 연동 구조

Android Java 애플리케이션에서 리눅스 기반의 FPGA 하드웨어 장치 제어를 위해 JNI(Java Native Interface)를 활용합니다. Java 코드에서 직접 C로 구현된 네이티브 함수들을 호출함으로써, 안드로이드 앱 상에서 외부 디바이스와의 직접적인 상호작용을 가능하게 합니다.

.

4.3.1 Java ←→ JNI 연동 구조

단계	구성 요소	설명
1	Java native method 선언	Java 클래스 내에 native 키워드를 사용하여 선언
2	.so 라이브러리 로드	System.loadLibrary("device-jni") 사용
3	JNI C 함수 구현	JNIEXPORT 키워드와 JNI naming 규칙에 따라 C 함수 정의
4	장치 파일 제어	/dev/fpga_*.c 장치 파일을 open()/write()/read() 방식으로 접근
5	Android.mk를 통한 빌드	JNI 코드를 .so로 컴파일 후 /system/lib/로 push
6	Java에서 함수 호출	Java 애플리케이션 내에서 네이티브 메서드 호출로 장치 제어 수행

4.3.2 JNI 함수별 설계

함수명 (JNI)	Java 메서드	장치 파일	기능 설명
led()	Device.led(String level)	/dev/fpga_led	왼쪽부터 n개의 LED를 점등
dip()	Device.dip()	/dev/fpga_dipsw	16비트 DIP 상태 문자열 반환
segment()	Device.segment(String val, String mode)	/dev/fpga_segment	정수 출력 또는 자동 카운팅
dotWrite()	Device.dotWrite(String level)	/dev/fpga_dotmatrix	숫자 1~16 대응 dot 표시
lcd()	Device.lcd(String text)	/dev/fpga_textlcd	LCD 문자열 출력
piezo()	Device.piezo(String soundId)	/dev/fpga_piezo	1바이트 사운드 ID 출력
getKeypad Code()	Device.getKeypadCode()	/dev/input/event4	키패드 입력값 (1~11) 반환

4.3.3 주요 처리 흐름

1) 모든 JNI 함수는 Java에서 받은 문자열 또는 숫자 인자를 C에서 적절한 형태로 변환 (GetStringUTFChars, atoi, write) 후 장치에 전달
2) 입력 장치 (keypad, dip)은 read() 함수로 값을 받아 Java로 반환
3) 출력 장치 (led, segment, dot, lcd, piezo)는 open() → write() → close() 구조로 동작

4) segment()의 경우 mode값이 1이면 0.05초 간격으로 자동 증가 표시 수행

5. 하드웨어 연동 설계 및 동작 구조

앱에서 각 하드웨어 장치를 어떻게 제어하는지, 각 함수가 어떤 JNI를 통해 어떤 디바이스 드라이버를 호출하고, 앱 내에서 어떻게 사용되는지 단계적으로 설명합니다.

5.1 하드웨어 제어 함수 설계 개요

각 하드웨어 장치별 Java ↔ JNI ↔ Device 드라이버 호출 구조

장치명	Java 함수 (Device.java)	JNI 함수명 (device-jni.c)	제어 파일 경로
LED	led(String ledValue)	Java_com_example_test5_Device_led	/dev/fpga_led
DIP 스위치	dip()	Java_com_example_test5_Device_dip	/dev/fpga_dipsw
7-Segment	segment(String val, String mode)	..._segment	/dev/fpga_segment
LCD	lcd(String text)	..._lcd	/dev/fpga_textlcd
Piezo	piezo(String musicType)	..._piezo	/dev/fpga_piezo
Dot Matrix	dotWrite(String pattern)	..._dotWrite	/dev/fpga_dotmatrix
Keypad	getKeypadCode()	..._getKeypadCode	/dev/input/event4

5.2 Java → JNI → Driver 흐름도

App (Activity) 사용자 입력에 따라 하드웨어 제어 요청을 전달

↓

HardwareManager.java 각 장치별 동작을 추상화하여 제어 함수 제공

↓

Device.java (native method) native 메서드 정의, JNI를 통해 C 코드 호출

↓

JNI C 코드 (device-jni.c) Java 요청을 디바이스 드라이버 호출로 변환

↓

Linux Driver (/dev/fpga_*) 실제 디바이스 접근을 처리하는 커널 모듈

↓

FPGA Hardware 입력 신호에 따라 실제 하드웨어 동작 수행

5.3 하드웨어 제어 함수별 작동 원리

항목별 작동 원

항목	설명
LCD (lcd)	최대 32자까지 문자열 출력. C 함수에서는 write()로 /dev/fpga_textlcd에 문자열 전달.
Segment (segment)	value: 6자리 숫자 문자열, mode: "0" 고정 출력, "1" 카운트업. write() 또는 ioctl() 사용.
LED (led)	"0"~"8": 왼쪽부터 점등, "-1"은 모든 LED 깜빡임. /dev/fpga_led에 write() 호출.
DIP (dip)	DIP 스위치 16비트 상태를 문자열로 반환. JNI에서 read()로 읽고 문자열로 변환.
Piezo (piezo)	"0" 정지, "1" 성공음, "2" 실패음. /dev/fpga_piezo에 write로 신호 전달.
Dot Matrix (dotWrite)	7x10 패턴 문자열 (예: "1", "2" 등) 을 전달. 반복출력을 위해 별도 dot 스레드 사용.
Keypad (getKeypadCode)	/dev/input/event4에서 blocking read(). JNI 내부에서 직접 polling 또는 select 사용.

5.4 앱 내 **HardwareManager** 역할

HardwareManager.java는 앱의 각 Activity 또는 게임 루틴에서 하드웨어 장치를 간접 제어하는 통합 인터페이스 클래스입니다. 하드웨어 제어 요청은 모두 이 클래스를 통해 이루어 집니다.

단순 출력형: LCD, Segment, LED, Piezo, Dot → Device.xxx(...) 직접 호출
지속형 출력: Dot → Thread를 사용하여 반복적으로 dotWrite() 호출
입력 감지형: Keypad → 내부 KeypadThread로 폴링 후 콜백 전달
조합형: DIP → dip() 호출 후, 가장 왼쪽의 1 bit 위치 계산 반환

하드웨어 제어 분류 유형

분류	설명	예시 메서드
단순 출력형	한 번의 함수 호출로 하드웨어가 즉시 반응하며 추가 상태 유지나 스레드 필요 없음	showLcdText(), showCountdown(), showStrikeCount(), playSuccessSound()
지속형 출력형	일정한 주기로 같은 출력이 반복되어야 하며, 자체 스레드로 제어	startDot(), stopDot()
입력 감지형	외부 입력을 감지해야 하므로, 별도의 스레드가 반복적으로 상태를 확인 (polling)	startKeypadListener(), stopKeypadListener()

조합형 (상태 분석)	단순 read 결과를 가공하여 의미 있는 값으로 변환하는 로직 포함	getHighestDipSwitchOn()

5.5 장치별 앱 작 단계 정리

각 장치에 대해 Activity → HardwareManager → Device(native) → JNI → C → 디바이스 드라이버 흐름으로 세분화하여 설명합니다.

5.5.1 LCD (/dev/fpga_textlcd)

1) Activity에서 HardwareManager.showLcdText("Ready") 호출
2) Device.lcd("Ready")로 전달됨 (native method)
3) JNI 통해 C 함수 호출 → open("/dev/fpga_textlcd") → write("Ready") → close()
4) LCD 화면에 문자열 "Ready" 출력됨

5.5.2 Segment (/dev/fpga_segment)

1) HardwareManager.showCountdown("333333") 또는 startSegmentCountUp() 호출
2) 각각 "value", "mode"를 인자로 Device.segment() 호출
3) JNI → C → write() or ioctl()로 /dev/fpga_segment 제어
4) 숫자 6자리 또는 카운트업 진행 (mode "1"일 경우 자동 증가)

5.5.3 LED (/dev/fpga_led)

1) HardwareManager.showStrikeCount(3) 호출
2) Device.led("3") → "3"은 왼쪽부터 3개 켜짐 의미
3) JNI → C → write("3")
4) LED 3개 점등됨

5.5.4 DIP Switch (/dev/fpga_dipsw)

1) HardwareManager.getHighestDipSwitchOn() 호출
2) Device.dip() 호출 → JNI에서 read() → "010000..." 문자열 반환
3) 왼쪽부터 탐색하여 가장 먼저 "1"인 위치를 반환 (i + 1)
4) 사용자는 DIP 스위치로 레벨 등을 선택 가능

5.5.5 Piezo (/dev/fpga_piezo)

1) playSuccessSound() 또는 playFailSound() 호출
2) Device.piezo("1") or "2" 호출 후, 1초 뒤 Device.piezo("0")로 종료
3) JNI → C → write("/dev/fpga_piezo")
4) 피에조 스피커에서 성공/실패음 재생됨

5.5.6 Dot Matrix (/dev/fpga_dotmatrix)

1) HardwareManager.startDot("2") 호출

2) 내부 dotThread 생성 → 1ms 주기로 Device.dotWrite("2") 반복 호출
3) JNI → write("/dev/fpga_dotmatrix")
4) 숫자 "2" 도트 출력 반복. stopDot() 호출 시 스레드 종료

5.5.7 Keypad (/dev/input/event4)

1) HardwareManager.startKeypadListener(callback) 호출
2) 내부 KeypadThread 실행 → 반복적으로 Device.getKeypadCode() 호출
3) JNI → C에서 read("/dev/input/event4")
4) 키 입력 발생 시, 콜백 callback.onKeyPressed(int keyCode) 호출

5.6 스레드 기반 장치 제어 설계

하드웨어 제어 중 일부 장치는 지속적인 출력을 반복하거나, 외부 입력을 비동기적으로 감지해야 하므로, 별도의 스레드 기반 처리가 필요하다. Dot Matrix 출력과 Keypad 입력 감지에서 이러한 방식이 적용되습니다.

5.6.1 Dot Matrix 출력 스레드 설계

목적	숫자 또는 패턴을 Dot Matrix에 지속적으로 출력
주요 클래스	Device.dot (내부 Thread 클래스)
동작 방식	Dot Matrix는 write 후 일정 시간 지나면 사라지기 때문에, 반복 출력 유지 필요
시작 함수	HardwareManager.startDot(String pattern) → 내부에서 new Device.dot(pattern).start() 실행
반복 제어	내부 run()에서 1ms 간격으로 dotWrite(pattern) 호출
정지 처리	HardwareManager.stopDot() → 내부 stopThread() 호출로 running = false 설정하여 종료
JNI 처리	dotWrite() → JNI → /dev/fpga_dotmatrix에 write

5.6.2 Keypad 입력 감지 스레드 설계

목적	키패드 이벤트를 비동기적으로 감지
주요 클래스	HardwareManager.KeypadThread (내부 Thread 클래스)
콜백 구조	startKeypadListener(KeypadCallback callback) 함수로 외부에서 콜백 등록
감지 방식	Device.getKeypadCode() JNI 호출 → /dev/input/event4에서 blocking read
반복 간격	50ms 간격으로 polling

키 처리	읽은 값이 있으면 callback.onKeyPressed(key) 호출로 앱에 전달
종료 처리	stopKeypadListener() 호출 시 interrupt()로 스레드 종료

6. 테스트 및 결과

6.1 기능별 테스트 체크리스트

항목	확인 방식	결과
Keypad 입력	dmesg 로그 확인 및 UI 반응 확인	정상 동작
LED 점등	단계별 점등 개수 시각 확인	정상 동작
LCD 출력	텍스트 표시 확인 (Ready, Go!, 결과 등)	정상 동작
7-Segment	숫자 출력 및 카운트업 확인	정상 동작
Dot Matrix	숫자 패턴 유지 출력 확인	정상 동작
DIP 스위치	ON 위치에 따라 레벨 설정 확인	정상 동작
Piezo 사운드	게임 성공/실패 시 소리 확인	정상 동작
진동 모터	실패 시 진동 발생 확인	정상 동작

6.2 시나리오 기반 테스트

테스트 시나리오

단계	동작 내용	검증 항목	결과
1	DIP 스위치를 3번째 위치 ON	DIP 상태 반영 → Level 3	정상
2	Keypad 11번으로 레벨 선택 확정	Dot Matrix에서 숫자 3 표시	정상
3	게임 시작 → 3초 카운트다운	LCD: “Ready” / “Go!”Segment: 333333→222222→111111	정상
4	Keypad 1~9로 입력 시도(오입력 포함)	오입력 시 LED 증가 + LCD: “Wrong”	정상
5	타임아웃 발생 시	LCD: “Miss”, LED 점등, strike 증가	정상
6	8회 이상 strike 발생	LED 깜빡임, LCD: “Fail”, Piezo: 실패음	정상
7	게임 종료 후 결과 화면 이동	LCD: 결과 메시지Segment: 최종 시간 정지Dot Matrix 유지	정상
8	Keypad 0/11로 결과 페이지 버튼 이동	버튼 간 포커스 이동 및 실행 확인	정상

7. 결론 및 느낀점

유준혁

프로젝트를 진행하면서 하드웨어와 소프트웨어의 연결에 대해 직접적으로 볼 수 있어 실감할 수 있었습니다. 특히 디바이스 드라이버와 JNI를 이용해 안드로이드 앱과 보드가 통신할 때 아주 많은 시행착오도 겪었습니다.

FPGA를사용하기 위해 디바이스 드라이버를 분석하면서 단순한 출력에도 복잡하고 많은 로직이 필요하다는 것을 느꼈습니다. 그 밖에도 C 코드와 자바 코드가 처음부터 연동을 염두해 두고 개발된 언어가 아니었을 텐데도 어떻게 연동될 수 있는 것인지 신기했습니다. 그리고 앱에서 멀티 스레드와 핸들러, C에서 while이 같이 돌면서 병렬적으로 동작하는 장치를 관리하는 것이 얼마나 복잡한지도 배웠습니다. 어려움이 많은 수업이었지만 그만큼 도전적이고 많은 것을 배울 수 있었던 수업이라고 생각합니다. 감사합니다.

김태경

이번 프로젝트를 통해 기존에 경험했던 단순한 안드로이드 앱 개발을 넘어, 임베디드 하드웨어 장치들과 직접 연동하는 새로운 분야를 경험할 수 있었습니다. 초기에는 SD카드 이미지 수정, 커널 업데이트, Hanback 전용 디바이스 드라이버 접근 등 생소한 환경 설정에 진도를 따라가기 바빴지만, 안드로이드 스튜디오 개발 환경이 정비되고 첫 번째 디바이스인 LED 출력이 성공하면서 조금씩 감을 잡을 수 있었습니다.

이후에는 리눅스 개발에 익숙한 팀원(유준혁)이 C 기반 드라이버 코드를 작성하고, 저는 앱 UI 및 동작 로직을 구현한 뒤 연동을 시도하는 방식으로 역할을 분담했습니다. 하지만 초기 설계 부족으로 I/O 연동 방식에 대한 명확한 기준 없이 진행한 탓에, 개별 기능은 완성되었음에도 앱과 디바이스를 제대로 연결하기 어려운 상황이 발생했습니다. 결국 기존의 코드들을 분석하고, JNI를 통한 정석적인 Java ↔ C 연동 방식을 적용하여 구조를 재정비하게 되었습니다. 특히 키패드의 스레드 설계에서는 입력 처리 방식에서 한 차례 더 어려움을 겪었고, 이 경험을 통해 시스템 설계의 중요성과 사전 구조화의 필요성을 한번 더 절실히 느꼈습니다.

우여곡절 끝에 대부분의 디바이스와의 연동을 성공적으로 구현하였고, 앱도 큰 문제 없이 작동하게 되었습니다. 비록 스레드 처리로 인해 앱이 간헐적으로 중지되는 문제가 있었지만, 전체적인 목표를 만족하며 프로젝트를 완성할 수 있었던 값진 경험이었습니다. 이번 과정을 통해 설계, 디버깅, JNI, 리눅스 디바이스 드라이버 등 실질적인 시스템 연동 역량을 크게 키울 수 있었습니다.

8. 참고자료

- HBE-SM5-S4210으로 배우는 안드로이드, (주) 한백 전자 기술 연구소

- Android NDK 공식 문서, Android Developers (https://developer.android.com/ndk)
- 커널 모듈과 디바이스 드라이버 개발 실습, FastTrack Embedded Systems Lab
- Android Developers JNI 가이드, Android Developer Docs (https://developer.android.com/training/articles/perf-jni)
- 프로젝트 중 사용된 소스코드, 드라이버 예제, JNI 연동 자료 및 실습 코드 일체(강의자료)