Embedded System Software 프로젝트

(프로젝트 수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20181625, 남현준

개발기간: 2024. 05. 20. -2024. 06. 04.

최 종 보 고 서

I. 개발 목표

- 각 과제마다 주어지는 주제를 바탕으로 본 과제에서 추구하는 개발 목표를 설정하고 그 내용을 기술할 것.

수업에서 배운 Android NDK를 이용해 프로그래밍하는 방법을 이해하고, 이해한 내용을 바탕으로 실습에서 사용하는 Achro-i.mx6q 보드에서의 구현을 진행한다. Native 키워드를 이용해 Java에서 특정 메소드를 호출하면 C로 정의된 코드가 수행될 수 있도록 JNI 코드를 정의하고 이를 이용해 Java 응용 프로그램이 FPGA 모듈을 적절하게 제어할 수 있도록 한다.

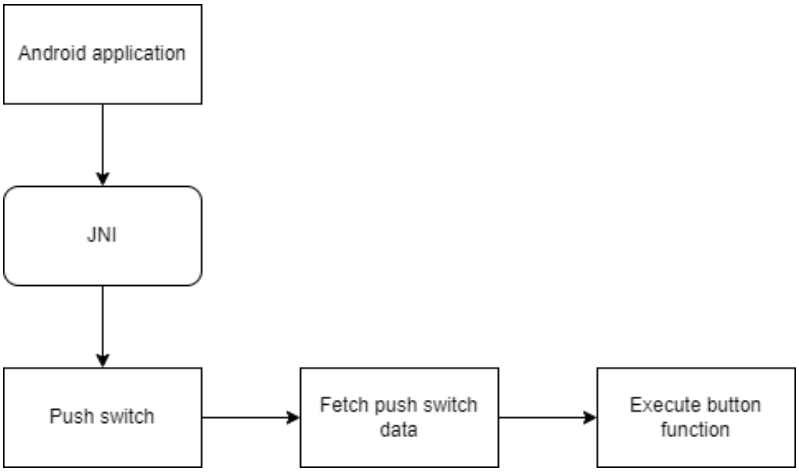
II. 개발 범위 및 내용

- 자신들이 설계한 개발 목표를 달성하기 위하여 어떠한 내용의 개발을 수행할 지 그 범위와 개발 내용을 기술할 것.

가. 개발 범위

- Application

  사용자의 손가락을 통해 입력을 감지하고, 움직임에 따라 화면 상에 결과를 반영하는 그림판 프로그램을 Android를 통해 구현한다. 이 때, 수행되는 결과에 따라 적절하게 FPGA 모듈을 갱신할 수 있도록 Java Native Interface (JNI)를 통해 각 모듈의 open(), read(), close() 등이 수행되도록 한다. 이를 flow chart로 표현하면 다음과 같다.



♦ Device driver

Android에서 구현한 그림판에서 손가락의 입력을 감지하고 JNI를 통해서 각 모듈에서의 값 갱신이 이뤄질 수 있어야 한다.

JNI를 통해서 FPGA 모듈을 제어할 때, FND에 현재까지 손가락 입력을 통해 화면 상에 무언가가 그려진 횟수, LCD의 첫 16글자에 현재 그리고 있는 모드의 종류와 두번째 16글자에 그려지는 시작-끝 좌표쌍, push switch를 통해 그림판에 redo, undo 등의 기능이 수행될 수 있도록 한다. FND에는 펜, 지우개 모드인 경우 획을 그리게 될 때의 stroke size가 어느 정도인 출력할 수 있도록 한다.

나. 개발 내용

♦ Android application

응용 프로그램에서 그림 프로그램이 실행되고 사용자의 손가락 움직임을 통해 화면 상에 그려질 수 있도록 프로그램을 구현한다. 이후, 프로그램의 상황에 따라 FPGA 모듈에 갱신이 이뤄질 수 있도록 JNI를 통해 device에 접근할 수 있도록 한다.

♦ 디바이스 드라이버

다음 FPGA 모듈들에 대해 android application이 수행되며 제어를 하게 된다.

- FND
- LCD
- PUSH SWITCH
- DOT

다음과 같은 조건을 만족하도록 구현한다:

- Android application이 처음 실행되는 경우, 초기 상태로 FND에 “0000”이 주어진다
- LCD의 초기 상태는 첫 16글자에 “Pen “, 나머지 16글자는 공백상태이다.
- DOT의 초기 상태로는 1 (stroke size)이 주어진다.
- PUSH SWITCH는 일정 시간마다 입력이 있었는지 확인하고 입력에 따라 다음과 같은 작동이 실행될 수 있어야 한다.
- (0)이 입력된 경우, 그림판에서의 undo, (1)이 입력된 경우 redo, (3)과 (4)의 경우 stroke 크기 조절, (8)의 경우 그림판에 clear를 수행시켜 초기 상태로 돌아갈 수 있도록 한다.

III. 개발 방법

가. 개발 방법

♦ 응용 프로그램

Android application에서는 JNI를 통해 FPGA device driver를 호출하여 해당 모듈들의 값을 갱신한다.

JNI를 통해 프로젝트에서 사용되는 FPGA 모듈에 접근하고 값을 갱신할 수 있도록 각 모듈의 open(), close() 함수를 JNI 형식으로 선언 및 구현한다.

```
public native int openSwitch();
public native int closeSwitch();
public native int readSwitch(int fd);
public native int openDot(int value);
public native int openFND(String str);
public native int openLCD(String str);
public native void closeLCD();
public native void closeFND();
public native void closeDOT();
public native void writeLCDFirstLine(String str);
public native void writeLCDSecondLine(int mode, int x1, int y1, int x2, int y2);
public native int closeDot(int fd);
public native void startPushSwitch();
```

JNI를 통해 선언한 함수를 android application에서 사용하기 위해서는 해당 내용을

ndk-build를 통해서 application에 build를 진행해야 한다. 수정한 부분이 있다면 Android.mk 파일에서 LOCAL_SRC_FILES 부분에 native 함수를 선언한 파일의 이름을 변경한다.

```
LOCAL_PATH:=$(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE:=ndk-exam
LOCAL_SRC_FILES:=first.c second.c
LOCAL_LDLIBS := -llog
#LOCAL_LDLIB := -L$(SYSROOT)/usr/lib -llog

include $(BUILD_SHARED_LIBRARY)
```

Application의 화면을 구성하기에 앞서 요소들을 화면에 추가하고 싶은 경우 layout에 해당 요소를 추가해야 후에 구현했을 때 화면 상에 출력된다. 이를 위해 /res/layout 디렉토리의 activity_main.xml 파일의 수정을 통해 해당 요소들이 선언되며 화면에 출력될 수 있도록 한다.

```
 <Button
     android:id="@+id/ColorPicker"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Color Picker" />
```

```
<Button
    android:id="@+id/Mode"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pen"
/>
```

```
<Button
android:id="@+id/undo"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="undo" />

<Button
android:id="@+id/redo"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="redo" />
```

각 버튼은 구현될 그림판 프로그램에서 색상 선택, 도형 선택, undo, redo 기능을

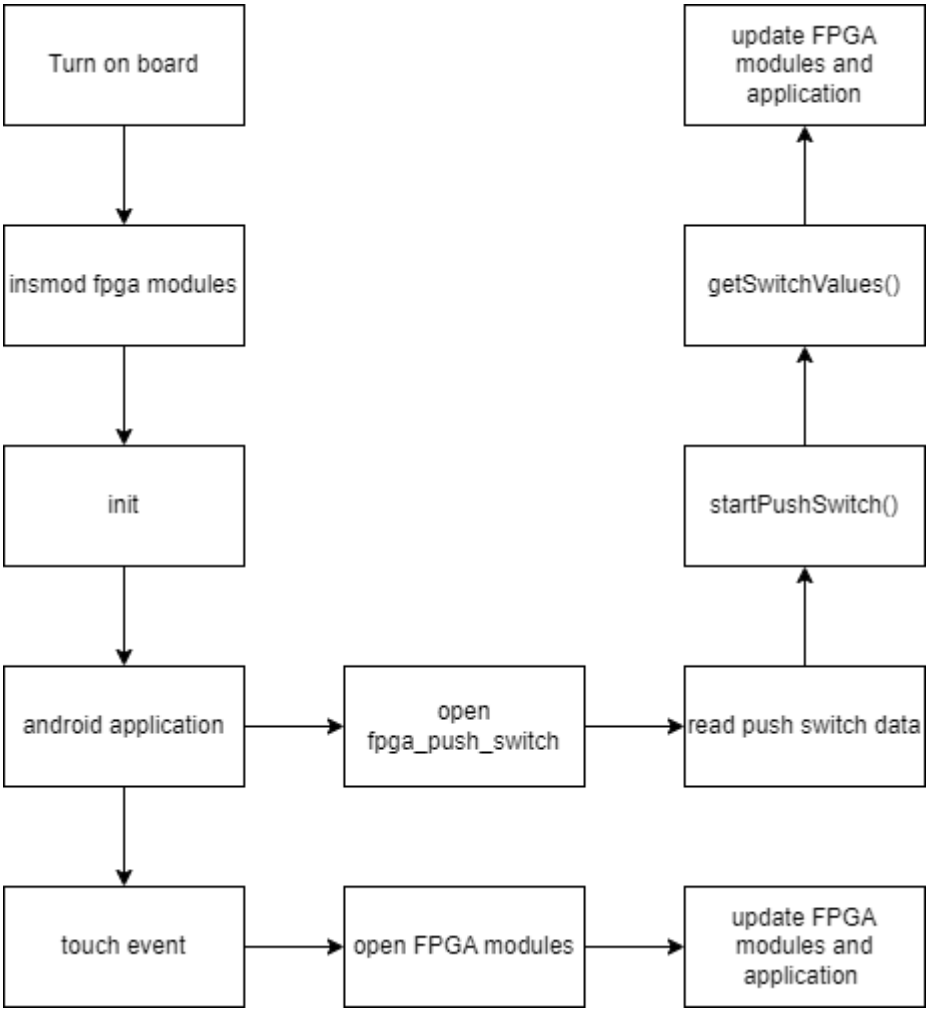
담당하게 될 것이다.

Application이 생성될 때, 화면의 구성과 필요한 요소들이 적절하게 배치될 수 있어야 한다. 이를 위해 onCreate()를 선언하고 수정을 통해 application 생성 시 프로그램에서 사용할 view의 정의와 그에 맞는 button의 추가를 진행한다.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.activity_main);

    paintView = new PaintView(this);

    ColorDisplay = (TextView)findViewById(R.id.colorDisplay);
    LinearLayout container = (LinearLayout)findViewById(R.id.container);
    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.MATCH_PARENT);
    container.addView(paintView, params);
    Button ColorPicker = (Button)findViewById(R.id.ColorPicker);
    Button Mode = (Button)findViewById(R.id.Mode);
    Button undo = (Button)findViewById(R.id.undo);
    Button redo = (Button)findViewById(R.id.redo);
    input = (TextView)findViewById(R.id.number);
    undo = (Button)findViewById(R.id.undo);
    redo = (Button)findViewById(R.id.redo);
    paintView.SetTextView(ColorDisplay, input);
    paintView.SetButtons(ColorPicker, undo, redo, Mode);
    paintView.SetButtonListeners();
```

해당 버튼들을 layout에서 선언한 변수의 위치에 맞게 생성될 수 있도록 findViewById()를 통해서 각 변수를 찾은 다음 선언하도록 한다. 이후, 해당 class에서 정의한 내용을 실제 실행되는 view인 paintView에서 버튼의 정의 및 클릭 시 실행되는 logic을 선언하도록 한다.

이렇게 실행되는 프로그램의 전체 흐름은 다음과 같다.



실행되는 view인 PaintView의 정의를 진행한다.

```
private void initPaintView() {
    mPaint = new Paint();
    mPaint.setColor(Color.BLACK);
    curColor = Color.BLACK;
    mPaint.setStyle(Paint.Style.FILL);
    n = 0;
    Mode = 1;
    mPaint.setAntiAlias(true); // Smooth drawing
    mPath = new Path();
}
```

View에서 생성될 버튼의 정의를 진행한다.

```
public void SetButtons(Button __colorPicker, Button __unDo, Button __reDo, Button __toggleMode) {
    colorPicker = __colorPicker;
    reDo = __reDo;
    unDo = __unDo;
    toggleMode = __toggleMode;
}
```

```
public void SetButtonListeners() {
    colorPicker.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            showColorPickerDialog();
        }
    });
```

```
unDo.setOnClickListener(new OnClickListener(){
     @Override
     public void onClick(View v) {
         if(DrawingEvents.size() == 0){
             invalidate();
             return;
         }
```

```
reDo.setOnClickListener(new OnClickListener(){
 @Override
    public void onClick(View v) {
        if(ReDos.size() == 0) return;
        Shapes tmp = new Shapes();
        tmp = ReDos.getLast();
        DrawingEvents.addLast(tmp);
        ReDos.removeLast();
```

```
toggleMode.setOnClickListener(new OnClickListener(){
     @Override
       public void onClick(View v) {
         Mode = (Mode + 1) % 2;
         if(Mode == 0)
             toggleMode.setText("Rectangle");
         else
            toggleMode.setText("Pen");
     }
});
```

생성되는 버튼들에 대해 OnClickListener()를 추가해주는 것으로 버튼 클릭시 실행되는 작동을 선언하고 해당 내용대로 실행될 수 있다.

색상 선택 버튼의 경우, 여러 개의 선택지 중 하나를 선택하고, 그에 따라 그리는 색상이 변할 수 있도록 구현하고자 했다. 이를 위해, dialog를 선언하여 선택할 수 있는 색상의 목록이 application의 화면에 나타날 수 있도록 했다.

```
private void showColorPickerDialog() {
    final int[] colors = new int[]{
            Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW,
            Color.CYAN, Color.MAGENTA, Color.BLACK, Color.WHITE
    };

    final String[] colorNames = new String[]{
            "Red", "Green", "Blue", "Yellow",
            "Cyan", "Magenta", "Black", "White"
    };

    AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
    builder.setTitle("Pick a color");
    builder.setItems(colorNames, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            mPaint.setColor(colors[which]);
            updateColorDisplay(colors[which], colorNames[which]);
            curColor = colors[which];
        }
    });
    builder.show();
}
```

Dialog에도 OnClickListener()를 통해 클릭 시 특정 작동이 수행될 수 있도록 구성한다.

Toggle mode의 경우, 클릭마다 사용하는 도구가 바뀔 수 있도록 설정했다.

```
toggleMode.setOnClickListener(new OnClickListener(){
    @Override
      public void onClick(View v) {
        Mode = (Mode + 1) % 4;
        if(Mode == 0){
            toggleMode.setText("Rectangle");
           curStrokeSize = 1;
           mPaint.setStrokeWidth(curStrokeSize);
        }
        else if(Mode == 1)
           toggleMode.setText("Pen");
        else if(Mode == 2){
            toggleMode.setText("Oval");
           curStrokeSize = 1;
           mPaint.setStrokeWidth(curStrokeSize);
        }
        else{
            toggleMode.setText("Eraser");
        }

        Log.d("%s\n", curColorName);
       // n이 변경될 때마다 리스너 호출
         if (strokeChangeListener != null) {
            strokeChangeListener.onNumberChanged(curStrokeSize);
           }
       if (modeChangeListener != null) {
           modeChangeListener.onNumberChanged(Mode);
       }
    }

});
```

Application의 빈 공간에서 사용자의 손가락 입력을 감지하기 위해서는 onTouchEvent()를 사용하고, 이를 통해 사용자의 손가락 움직임을 관리할 수 있다.

MotionEvent를 통해서 touch event 발생 시 화면의 터치, 터치된 상태로 이동, 터치 종료를 식별할 수 있다. 각각 ACTION_DOWN, ACTION_MOVE, ACTION_UP으로 각 case에 대해서 처리할 수 있는 switch문을 선언했다.

```
switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        if(undoStack.isEmpty()){
            Bitmap bitmapCopy = Bitmap.createBitmap(mBitmap.getWidth(), mBitmap.getHeight(), mBitmap.getConfig());
            Canvas canvas = new Canvas(bitmapCopy);
            canvas.drawBitmap(mBitmap, 0, 0, null);

            // Push the copied bitmap to the undo stack
            undoStack.push(bitmapCopy);
        }
        startX = event.getX();
        startY = event.getY();

        if(Mode == 3){
            mPaint.setColor(Color.WHITE);
        }
        else{
            mPaint.setColor(curColor);
        }

        if (Mode == 1 || Mode == 3) {
            isDrawingLine = true;
            mPaint.setStyle(Paint.Style.STROKE);
            if (prevX == -99999999 && prevY == -99999999) {
                mPath.reset(); // Reset path for new touch
                mPath.moveTo(startX, startY); // Move to the starting point
                mX = startX;
                mY = startY;
            }
        }
        invalidate();
        break;
```

```
case MotionEvent.ACTION_MOVE:
    endX = event.getX();
    endY = event.getY();
    if(endX >1000)
        endX = 999;
    if(endY > 1000)
        endY = 999;
    if (Mode == 1 || Mode == 3) {
        isDrawingLine = true;
        float dx = Math.abs(endX - mX);
        float dy = Math.abs(endY - mY);
        if (dx >= 4 || dy >= 4) {
            mPath.lineTo(mX, mY); // Smooth curve
            mX = endX;
            mY = endY;
        }
    } else if(Mode == 0) {
        drawRectangle();
    }
    else
        drawOval();
    invalidate();
    break;
```

```
case MotionEvent.ACTION_UP:
    endX = event.getX();
    endY = event.getY();
    if (Mode == 0) {
        drawRectangle();
    } else if (Mode == 1 || Mode == 3) {
        isDrawingLine = true;
        mPath.lineTo(mX, mY); // Connect the path to the last point
        mCanvas.drawPath(mPath, mPaint); // Draw current path onto the canvas (bitmap)
        mPath.reset(); // Reset path
        mPaint.setStyle(Paint.Style.FILL);
    }
    else
        drawOval();
    invalidate();

    // 비트맵 복사 및 저장
    saveBitmap();
    Log.d("drawing", "DrawEvent Values: " + startX + " " + startY + " " + endX + " " + endY);
    Log.d("curColorName", curColorName);
    // n이 변경될 때마다 리스너 호출
    if (numberChangeListener != null) {
        numberChangeListener.onNumberChanged(n);
    }
    if (coordinatesChangeListener != null) {
        coordinatesChangeListener.onCoordinatesChanged(startX, startY, endX, endY);
    }
    break;
```

이 중, ACTION_UP에서는 한 번의 입력이 종료되는 시점이다. 이 경우, undo와 redo를 위해서 입력에 대한 정보를 저장할 필요가 있고, 현재 bitmap의 정보를 선언한 자료구조에 저장한다. 그 후, 그려진 내용을 화면 상에 반영하기 위해 invalidate()를 사용하여 화면에 갱신되어야 하는 부분의 갱신을 진행한다.

한편, View에서 특정 값에 갱신이 일어나는 경우, 해당 값을 반영할 수 있어야 한다. 이를 위해, 사용되는 변수별로 changeListener를 추가하여 각 변수에 변화가 생길 때마다 특정 작동을 할 수 있도록 설정한다.

```java
// 리스너 설정
paintView.setOnNumberChangeListener(new PaintView.OnNumberChangeListener() {
    @Override
    public void onNumberChanged(int newNumber) {
        String s;
        if(newNumber < 10){
            s = "000";
        }
        else if(newNumber < 100)
            s = "00";
        else if(newNumber < 1000)
            s="0";
        else
            s="";
        openFND(s + newNumber);
    }
});

// 리스너 설정
paintView.setOnStrokeChangeListener(new PaintView.OnNumberChangeListener() {
    @Override
    public void onNumberChanged(int newNumber) {
        openDot(newNumber);
    }
});

리스너 설정
paintView.setOnModeChangeListener(new PaintView.OnNumberChangeListener() {
    @Override
    public void onNumberChanged(int newNumber) {
        writeLCDFirstLine(modeArr[newNumber]);
        curMode = newNumber;
        if(curMode == 1 || curMode == 3){
            openDot(1);
        }
        else
            openDot(-1);
    }
});


paintView.setOnCoordinatesChangeListener(new PaintView.OnCoordinatesChangeListener() {
    @Override
    public void onCoordinatesChanged(float x1, float y1, float x2, float y2) {
        writeLCDSecondLine(curMode, Math.round(x1), Math.round(y1), Math.round(x2), Math.round(y2));
    }
});
```

해당 값들은 touch가 끝나거나 그림판에서 주요한 변화가 생길 때 갱신되는 값들로 해당 상황에 맞춰 FPGA 모듈의 값에도 갱신이 일어나야 한다. 각 Listener에서는 의도한 FPGA 모듈에 맞는 native 함수를 호출하여 모듈의 값에 갱신이 진행될 수 있도록 한다.

ChangeNumberListener는 현재까지 undo와 redo를 하지 않고 그림판에 그림을 그린 횟수를 나타내며 onTouchEvent()의 ACTION_UP이 실행될 때 n의 값이 갱신되며 실행되는 listener이다. 해당 값을 FPGA 상에 갱신할 수 있도록 다음과 같은

JNI native로 fpga_fnd 모듈에 접근하여 값을 수정한다.

```
JNIEXPORT jint JNICALL Java_org_example_ndk_CSE4116_1Final_openFND
  (JNIEnv *env, jobject this, jstring str){
    int fd =  open("/dev/fpga_fnd", O_RDWR);
    const char *nativeString = (*env)->GetStringUTFChars(env, str, 0);
    int str_size = strlen(nativeString);
    char fnd_str[5];

    int i = 0;

     if (strcmp(nativeString, "0000") == 0) {
            // If the input is "0000", set all to '\0'
            memset(fnd_str, '\0', 4);
        } else {
            for (i = 0; i < 4; i++) {
                fnd_str[i] = nativeString[i];
            }
        }
     for(i=0;i<str_size;i++)
         {
             if((nativeString[i]<0x30)||(nativeString[i])>0x39) {
                 printf("Error! Invalid Value!\n");
                 return -1;
             }
             fnd_str[i]=nativeString[i]-0x30;
         }

    write(fd,fnd_str,4);
    close(fd);

    (*env)->ReleaseStringUTFChars(env, str, nativeString);
}
```

setOnModeChangeListener는 그림판 상에 있는 mode 변경 버튼을 눌렀을 때 값이 갱신되며 현재 사용하는 도구의 종류를 LCD의 첫 16번째 글자에 나타나도록 한다. 이를 위해 다음과 같은 JNI native 함수를 이용해 fpga_text_lcd 모듈에 접근하고 값을 갱신할 수 있도록 한다.

```
JNIEXPORT jint JNICALL Java_org_example_ndk_CSE4116_1Final_openDot
  (JNIEnv *env, jobject this, jint value){
    int fd =  open("/dev/fpga_dot", O_WRONLY);
    LOGV("openDot value : %d\n", value);

    if(value == -1)
        write(fd, fpga_set_blank, strlen(fpga_set_blank));
    else{
        int str_size = sizeof(fpga_number[value]);
        write(fd,fpga_number[value],str_size);
    }
    close(fd);
}
```

```
JNIEXPORT jint JNICALL Java_org_example_ndk_CSE4116_1Final_openLCD
  (JNIEnv *env, jobject this, jstring str){
    int fd =  open("/dev/fpga_text_lcd", O_WRONLY);
    const char *nativeString = (*env)->GetStringUTFChars(env, str, 0);
    int str_size = strlen(nativeString);
    int i = 0;

//  for(i = 0; i < str_size; i++)
//      str_tmp[i] = nativeString[i];
//  memset(str_tmp+str_size, ' ', 32-str_size);
    LOGV("str_tmp in openLCD : %s\n", str_tmp);
    write(fd,nativeString,str_size);
    close(fd);
}
```

SetOnStrokeChangeListener는 현재 사용 중인 도구가 펜이거나 지우개일 때, 그려지는 굵기 값을 갱신하여 DOT상에 나타날 수 있도록 하는 listener이다. 따라서, 해당 listener가 실행되는 경우, fpga_dot에 접근할 수 있도록 다음과 같은 JNI native 함수를 이용한다.

```
JNIEXPORT jint JNICALL Java_org_example_ndk_CSE4116_1Final_openSwitch
  (JNIEnv *env, jobject this){
    if(dv)
        return -1;
    int dev = open("/dev/drivers", O_RDWR);
    if (dev<0){
            LOGV("Device Open Error\n");
            close(dev);
            return -1;
    }
    LOGV("openSwitch returned: %d" , dev);
    dv = dev;
    return dev;
}
```

이외에도 도구를 이용해 그림을 그렸다면 시작 좌표와 끝 좌표를 LCD의 두 번째

16개의 글자에 나타날 수 있도록 했다.

```
paintView.setOnCoordinatesChangeListener(new PaintView.OnCoordinatesChangeListener() {
    @Override
    public void onCoordinatesChanged(float x1, float y1, float x2, float y2) {
        writeLCDSecondLine(curMode, Math.round(x1), Math.round(y1), Math.round(x2), Math.round(y2));
    }
});
```

```
JNIEXPORT void JNICALL Java_org_example_ndk_CSE4116_1Final_writeLCDSecondLine
  (JNIEnv *env, jobject this, jint mode, jint x1, jint y1, jint x2, jint y2){
        int fd =  open("/dev/fpga_text_lcd", O_WRONLY);
        char lcd_tmp[16];

        LOGV("str_tmp before : %s\n", str_tmp);
        int i = 0;
        sprintf(lcd_tmp, "%03d %03d %03d %03d ", x1, y1, x2, y2);
        LOGV("str_tmp after : %s\n", lcd_tmp);

        if(mode == 0){
            char lcd_first[16] = "Rectangle       ";
            memcpy(&str_tmp[0], lcd_first, 16);
        }
        else if(mode == 1){
                    char lcd_first[16] = "Pen             ";
                    memcpy(&str_tmp[0], lcd_first, 16);
                }
        if(mode == 2){
                    char lcd_first[16] = "Oval            ";
                    memcpy(&str_tmp[0], lcd_first, 16);
                }
        if(mode == 3){
                    char lcd_first[16] = "Eraser          ";
                    memcpy(&str_tmp[0], lcd_first, 16);
                }
        memcpy(&str_tmp[16], lcd_tmp, 16);
        str_tmp[33] = '\0';
        write(fd,str_tmp, 32);
        LOGV("%s\n", str_tmp);
        close(fd);
}
```

PUSH SWITCH는 android application 실행 후, 일정 시간마다 push switch를 통해 입력된 값이 있는지 확인할 수 있어야 한다. 하지만, 그냥 입력값을 받아오도록 하는 경우, 프로그램이 비정상적으로 종료될 수 있기에 thread를 생성하고 해당 thread에서 push switch의 입력값을 받아 android application에서 사용할 수 있도록 한다.

```java
//push switch에 입력이 주어지는지 일정 시간마다 체크하기 위해 개별 thread로 관리.
new Thread(new Runnable() {
    @Override
    public void run() {
        startPushSwitch();
    }
}).start();
```

```java
public native void startPushSwitch();
```

```c
JNIEXPORT void JNICALL Java_org_example_ndk_CSE4116_1Final_startPushSwitch
  (JNIEnv *env, jobject this){
    int i;
    int dev;
    int buff_size;
    jclass switch_class;
    jmethodID methodID;

    unsigned char push_sw_buff[MAX_BUTTON];

    dev = open("/dev/fpga_push_switch", O_RDWR);

    if (dev < 0) {
            printf("Device Open Error\n");
            close(dev);
            return;
    }
    buff_size = sizeof(push_sw_buff);
    switch_class = (*env)->GetObjectClass(env, this);
    methodID = (*env)->GetMethodID(env, switch_class, "getSwitchValues", "([I)V");

    while(!quit){
        usleep(400000);
                read(dev, &push_sw_buff, buff_size);

                jintArray values = (*env)->NewIntArray(env, MAX_BUTTON);
                jint *valuesBody = (*env)->GetIntArrayElements(env, values, 0);

                for (i = 0; i < MAX_BUTTON; i++) {
                    valuesBody[i] = push_sw_buff[i];
                }

                (*env)->ReleaseIntArrayElements(env, values, valuesBody, 0);

                // Call the Java callback method
                (*env)->CallVoidMethod(env, this, methodID, values);

                (*env)->DeleteLocalRef(env, values);
    }
}
```

한편, 이렇게 startPushSwitch()를 통해서 push switch의 값을 일정 시간마다 얻어오게 되는데 이렇게 얻어오는 값을 곧바로 android application에서 사용하는 것은

불가능하다. Android에서 사용하기 위해서는 JNI interface를 통해서 native C로 얻은 값을 android에서 사용할 수 있는 형태로 변경해줄 필요가 있다. GetObjectClass(), GetMethodId() 등을 이용해 사용하는 class와 method를 특정하고 jintArray와 jint로 선언한 변수들을 통해 push_switch에서 읽은 값을 저장해 java에서 사용하는 data의 형식으로 변환한다. 이후, 사용이 끝난 변수들의 memory 할당을 해제해주고 native에서 사용하고자 했던 android application의 method를 호출하는 것으로 native C -> android application으로의 데이터 변환 및 사용이 가능하게 된다.

IV. 연구 결과

- 최종 연구 개발 결과를 자유롭게 기술할 것.

Java로 작성되는 android application에서 C로 작성한 함수들을 Java Native Interface를 통해 연결해서 사용하는 방법에 대해 배울 수 있었다.

Android application에서 push switch 입력을 구현하는 부분에서 어려움이 있었는데 기존에는 한 번의 입력을 받고 난 이후 application이 멈추게 되었다. 해당 문제는 push switch 입력을 일정 시간마다 받도록 되어 있어 android application이 실행되고 있는 main thread가 무한 루프 상태에 빠지게 되어 발생한 문제였다.

해당 문제의 해결을 위해 background에서 thread에서 작업을 수행하고 main UI에 업데이트 하는 방법을 배울 수 있었다.

V. 기타

- 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술할 것. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

한 학기 동안 배웠던 지식을 바탕으로 application의 구상 및 작동 간 device control을 생각해보고 구현해볼 수 있어서 좋았다.