SFO15-505: Introducing I2C and GPIO userspace APIs for Android

Satish Patel
Scope

- Background
- High level diagram
- API reference
- Why native libs?
- Why SDK Addon?
- Application reference
- Demo!!
Background

- Modularity spreading across all domains
  - e.g. Project ARA based on concept - Phoneblocks

- Plenty i2c/spi/gpio based sensors/devices..
  - But no straight APIs to Android at present

- It’s a kick start to build bridge to access non-conventional protocols inside Android
High Level Diagram - I2C

- **I2C APP**
  - I2C SDK Addon lib
  - I2cDevice
  - I2cManager

- **I2C NATIVE API** (libandroidruntime.so)
  - libudev

- **I2C HAL**
  - ioctl
  - /dev/i2c-x

- **APPLICATION**
  - II2cManagerService
  - I2cService

- **FRAMEWORK**
  - Read app meta-data and extra device information for which access is requested

- **KERNEL**
  - HAL

- **HARDWARE**
  - I2C DEVICE
High Level Diagram - GPIO

- **APPLICATION**
  - GPIO APP
  - GPIO SDK Addon lib

- **FRAMEWORK**
  - GpioManager
  - IGpioManagerService
  - GpioService

- **HAL**
  - libudev
  - GPIO HAL

- **KERNEL**
  - /sys/class/gpio

- **HARDWARE**
  - GPIO DEVICE
# API Reference - I2C

<table>
<thead>
<tr>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I2cManager</strong></td>
<td></td>
</tr>
<tr>
<td>getInstance(Context context)</td>
<td>Returns a new instance of this class</td>
</tr>
<tr>
<td>openDevice()</td>
<td>Opens an i2c device so it can be used to send and receive data using I2cDevice</td>
</tr>
<tr>
<td>closePfd()</td>
<td>Close ParcelFileDescriptor attached to i2c device</td>
</tr>
<tr>
<td>doTransaction(I2cMessage... msgs)</td>
<td>Perform i2c transaction</td>
</tr>
<tr>
<td>setSlave(int addr)</td>
<td>Set i2c slave address</td>
</tr>
<tr>
<td>getPfd()</td>
<td>Returns ParcelFileDescriptor of an i2c device</td>
</tr>
</tbody>
</table>
# API Reference - GPIO

<table>
<thead>
<tr>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GpioManager :</strong> getInstance(Context context)</td>
<td>Returns a new instance of this class</td>
</tr>
<tr>
<td><strong>GpioManager :</strong> openGpio(int gpio, java.lang.String direction)</td>
<td>Returns file descriptor of gpio device value</td>
</tr>
<tr>
<td>closeGpio(int gpio)</td>
<td>Close mentioned gpio number</td>
</tr>
</tbody>
</table>
Why libudev?

- Hotplug does not guarantee static device node
- I2C HAL opens device based on “product id” not with actual device node
- HAL finds correct i2c device based on “product id” and map the device node.

So HAL need libudev help!!!

```
shell@pxa1928:/ # cat /sys/class/i2c-dev/i2c-0/device/0-0030/name
88pm800
```
Why Native Libs?

- I2cDevice class has PFD:ParcelFileDescriptor
- JAVA developers can use - i/o stream to do transaction with i2c device using PFD
- Still “I want more…”
  - set slave address
  - multiple i2c transactions
  - setting 10 bit address scheme
Why Native Libs? Contd..

I2cService

- IPC latency
- Some apps need continuous data from the device might suffer

Native Extension (I2cDevice)
- Runs in application context
- APIs
  - setSlave()
  - doTransaction()
- Work on opened PFD
- No special permission needed
Why SDK Addon?

- Enable device specific features to app developers
- Avoid distribution of whole OEM specific SDK - Can run on existing AOSP SDK - Size matters !!!
- Only Stubs will be visible to App developer
- Easy of integration - Android Studio
<?xml version="1.0" encoding="utf-8"?>
<manifest ...
<uses-feature android:name="android.hardware.i2c" />
<application>
<uses-library android:name="com.google.ara.i2c" />
<activity.......>
<intent-filter>
<category android:name="android.intent.category.DEFAULT" />
<action android:name="android.hardware.i2c.action.I2C_DEVICE_ATTACHED" />
</intent-filter>
<meta-data android:name="android.hardware.i2c.action.I2CDEVICE_ATTACHED" android:resource="@xml/i2c_filter" />
</activity>
</application>
</manifest>

res/xml/i2c_filter.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<i2c-device product-id="<ABC>" vendor-id="<XYZ>" version="1.0"/>
</resources>
Application Front - open device & set slave

import com.google.ara.i2c.I2cManager;
import com.google.ara.i2c.I2cDevice;
import com.google.ara.i2c.I2cMessage;

......
......
private I2cManager i2cm;
I2cDevice mDevice;

void openI2cDevice() {
    i2cm = I2cManager.getInstance(this);
    mDevice = i2cm.openDevice();
    ......
    ......
}

// set slave addr

I2cDevice mDevice;
/* open i2c device before setting slave address */
public void setSlave() {
    int ret;
    ret = mDevice.setSlave(0x4c); //0x4c is for ref#
    if(ret == 0)
        Log.d(TAG, "Slave address set!!");
    else
        Log.d(TAG, "Error in setting slave address");
}
Application Front - i2c transactions

// use native APIs
public void readValueNative() {

    byte[] reg = new byte[1];
    byte[] data = new byte[1];
    int ret = 0;
    reg[0] = (byte)0x3A;

    /* Two operation is necessary for read operation
    * first write the register we want to read
    * and then read the data
    */
    I2cMessage[] msg = new I2cMessage[2];
    msg[0] = new I2cMessage(0x30, 0, 1, reg); /* slave addr, write, length, reg */
    msg[1] = new I2cMessage(0x30, 1, 1, data); /* slave addr, read, lenght, data */
    ret = mDevice.doTransaction(msg);
    String hexString = Integer.toHexString(data[0]);
    Log.d(TAG, "Read Value Native, hex string:" + hexString);
}

public void openI2cDevice() {

mDevice = i2cm.openDevice();
mPfd = mDevice.getPfd();

// open file stream using fd
mFile = pfd.getFileDescriptor();

mInputStream = new FileInputStream(pfile);
//reading the value from i2c device
mOutputStream = new FileOutputStream(pfile);

mInputStream = new FileInputStream(pfile);
}

byte[] data = {0,0,0,0,0};
byte[] cmd = new byte[1];
cmd[0] = (byte) 0x3A;
setSlave();

try {
    mOutputStream.write(cmd);
    mOutputStream.flush();
    mInputStream.read(data);
} catch (IOException e) {
    Log.d(TAG, "Something wrong with device read and write");
}
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-feature android:name="android.hardware.gpio"/>
</manifest>

<application...>
    <uses-library android:name="com.google.ara.gpio"/>
    <activity.....>
        <meta-data android:name="android.hardware.gpio.action.GPIO_DEVICE_ATTACHED"
            android:resource="@xml/gpio_filter"/>
    </activity>
</application>

res/xml/gpio_filter.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <gpio-device product-id="<ABC>" vendor-id="<XYZ>" version="1.0"/>
</resources>
References

● Helium - PXA1928 based release with i2c/gpio support available at
  http://releases-ara-mdk.linaro.org/ara_pxa1928-5.1.1-open/

● Release includes prebuilt binary, demo application binary and sources
Demo !!
Thank You