AN4503
应用笔记

环境传感器：用于 Android 的硬件抽象层

作者：Adalberto Muhuho / Lorenzo Sarchi

前言

本应用笔记为将意法半导体环境传感器 （气压、湿度、紫外线传感器）成功集成到 Linux/Android 操作系统提供指南。

目录

1 文档概述

本文档描述如何将意法半导体环境传感器集成到 Linux/Android 系统。

它提供关于如何管理此任务的详细信息和流程。

您可以通过当地销售代表获取文档中提及的 ST 代码。

为了成功集成不同类型的传感器，除了问题和可能的解决方案，还将讨论传感器 HAL （硬件抽象层）的配置文件。

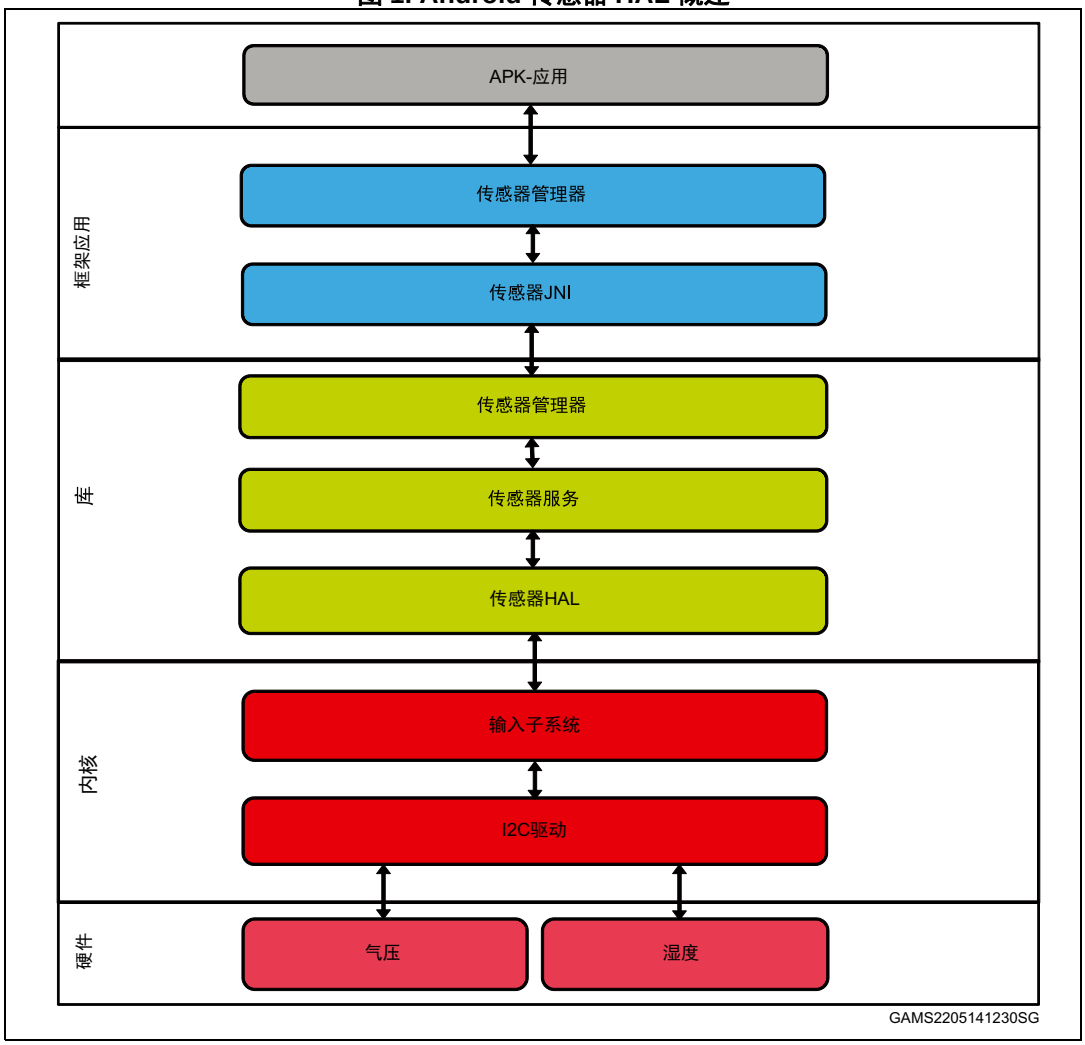
最后，还将描述该库的编译和安装。

1.1 Android 传感器 HAL 概述

Android 传感器 HAL 是提供从内核空间驱动程序到 Android 传感器服务和 Android 传感器管理器的链路的库。

Android 传感器框架的架构如下图所示。

图 1. Android 传感器 HAL 概述



1.1.1 Kernel

该层包含使用输入子系统 （所有输入设备的通用 Linux 框架）创建的 Linux 设备驱动。数据通过 Sysfs 虚拟文件系统导出到用户空间 （/sys/class/input/）。驱动通过稳定的 Linux 子系统 I²C 从 / 向传感器接收 / 发送数据。

1.1.2 传感器库

这些库用于为上层创建复杂接口。通过 SensorManager 类、传感器服务类和传感器 HAL 完成此任务。

1.1.3 应用框架

apk 应用使用该层从传感器获取数据。通信从创建传感器服务实例的 SensorManager 类开始，通过传感器 JNI （Java 本地接口）到达下层。

2 测试环境 / 生态环境

本文档适用于下列测试环境：

- Panda board：
  - 处理器：Omap4430
  - 测试板：PandaBoard ES Rev B2.
- 主机：
  - HP EliteBook 8470p
- Linux：
  - Ubuntu 13.04
- Android：
  - KitKat-4.4
- 编译环境：
  - androidearm-eabi7

2.1 将传感器连接到 PandaBoard

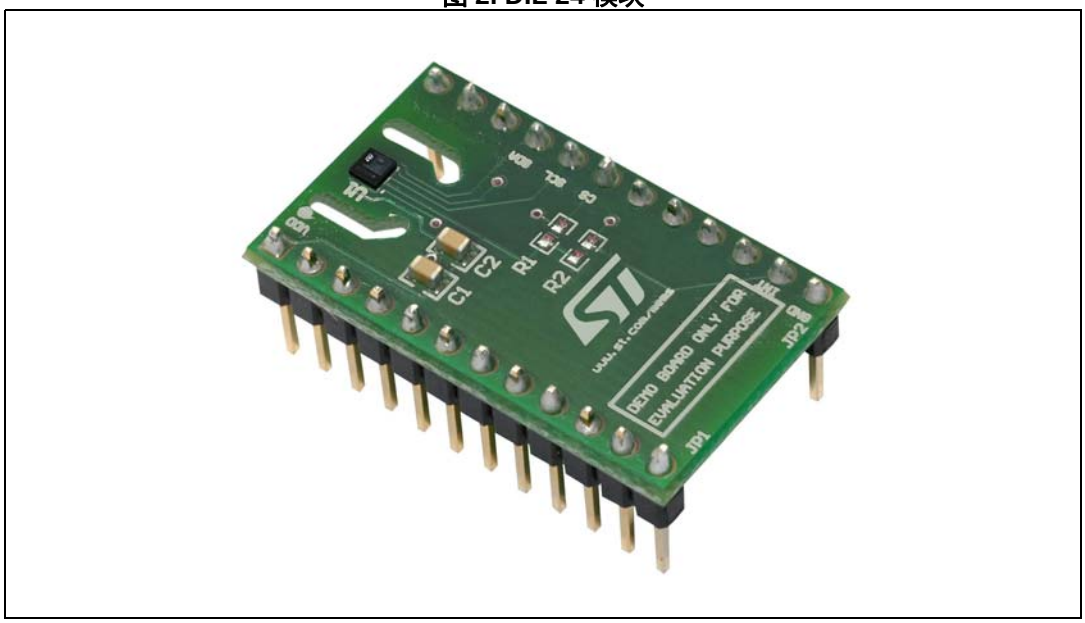
我们的意法半导体 DIL24 适配器 STEVAL-MKI141V2、 STEVAL-MKI142V1 和 STEVAL-MKI165V1 分别使用 “HTS221 湿度传感器 ”、“LPS25H 气压传感器 ” 和 “LPS25HB 气压传感器 ” 进行测试。

更多信息，请参考 www.st.com。

将适配器连接到 PandaBoard 的 J3 扩展连接器；使用 I²C 总线执行测试。

举例来说，对于下图中的 STEVAL-MK141V2 （可以对另外两块适配器重复相同步骤），假设适配器的脚位如下：

图 2. DIL 24 模块



引脚 1：Vdd ；引脚 2：Vdd_IO ；引脚 19：CS ；引脚 20：SCL ；引脚 21：SDA ；引脚 22：SDO

其中引脚 1 在左上，引脚 12 在左下，引脚 13 在右下，引脚 24 在右上。

这些引脚必须连接到 PandaBoard 的 “ 扩展连接器 A， J3” 的下列引脚：

表 1. DIL 24 模块对比 PandaBoard 连接

ST DIL24		PANDA J3 扩展连接器 A	
引脚编号	信号	引脚编号	信号
1	VDD	1	VIO_1V8
2	Vdd_IO	1	VIO_1V8
13	GND	28	GND
14	INT1	20	GPIO_134
19	CS (1)		
20	SCL	24	SCL
21	SDA	23	SDA
22	SDO	22	SDO/GPIO_39

1. CS 引脚必须连接至 Vdd_IO

2.2 Ubuntu 13.04 环境的特定设置

在完成 Ubuntu-13.04 的标准安装后，应用一些特定设置。

使用的软件包版本：

- Java：JDK1.6.0_45 和 JRE1.6.0_45
- GNU Make 3.82
- Python 2.7.4

下列网页描述了其他安装软件包：

“http://source.android.com/source/initializing.html#installing-required-packages-ubuntu-1204”，其中用 amd64 替代了 i386 标签用于 64 位软件包。

注： *在编写时， 1304 软件包的特有副标题不存在。*

- sudo apt-get install git gnupg flex bison gperf build-essential \
  zip curl libc6-dev libncurses5-dev:amd64 x11proto-core-dev \
  libx11-dev:amd64 libreadline6-dev:amd64 libgl1-mesa-glx:amd64 \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev: amd64
- sudo ln -s /usr/lib/amd64-linux-gnu/mesa/libGL.so.1 /usr/lib/ amd64-linux-gnu/libGL.so

用相应的 32 位 libz 替代 64 位 libz.so.1。

- sudo apt-get install lib32z1.

在使用代理时，为了使用上述及其他设置所需的 “apt-get”，使用下列单行代码创建文件 /etc/apt/apt.conf.d/01proxy：

- acquire: http::Proxy http://username:password@proxyname:8080

使用下列单行代码在文件夹 “/home/user” 中创建文件 bashrc：

- export USE_CACHE=1

从 Android Root 源码应用下列指令：

[RAS]: /prebuilts/misc/linux-x86/ccache/ccache -M 50G

为了下载和编译内核源码，将 ‘git’ 进行如下配置：

- $ git config --global user.email "e-mailing_address"
- $ git config --global user.name "*user*"

为了使用 fastboot 指令和 adb 指令连接到测试板，将下列代码行添加到文件 /etc/udev/rules.d/51-android.rules。

- # adb protocol on panda (PandaBoard)
- SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d101", MODE="0666", owner=”username”
- # fastboot protocol on panda (PandaBoard)
- SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d022", MODE="0666", owner=”username”
- # usbboot protocol on panda (PandaBoard)
- SUBSYSTEM=="usb", ATTR{idVendor}=="0451", ATTR{idProduct}=="d010", MODE="0666", owner=”username”

还必须修改该文件的权限：

"chmod a+x /etc/udev/rules.d/51-android.rules.

然后，使用指令 “sudo service udev restart” 重启 udev 服务。

2.3 构建 Android KitKat-4.4

为了构建环境，需执行下列步骤：

- 下载软件包
- 使用需要的补丁
- 编译源码
- 烧写 image 文件到 PandaBoard SD 卡

下面几节将详细描述这些步骤。

2.3.1 下载软件包

表 2. 下载封装 （分步说明）

步骤	说明
在根目录中创建工作区	$ mkdir ~/panda_work
	$ export PANDA_WORK=~/panda_work
	$ mkdir ~/panda_work/android
	$ export ANDROID_ROOT=~/panda_work/android (1)
下载 Android 4.4 （可能需要几个小时）	$ cd $ANDROID_ROOT
	$ repo init -u https://android.googlesource.com/platform/manifest -b android-4.4_r1.1
	$ repo sync
为 PandaBoard 下载合适的图形二进制文件，用于内核中的 PVR 驱动	$ wget https://dl.google.com/dl/android/aosp/imgtec-panda-20130603-539d1ac3.tgz (2)
	$ tar zxvf imgtec-panda-20130603-539d1ac3.tgz
	$ ./extract-imgtec-panda.sh
在下载的 Android 4.4 源码中添加对 PandaBoard 的支持	$ cd $ANDROID_ROOT
	$ git clone https://github.com/sola-dolphin1/sola_device_ti_panda.git -b kitkat device/ti/panda (2)
下载工具链用于编译 x-loader、 u-boot 和内核	$ cd $PANDA_WORK
	$ git clone https://android.googlesource.com/platform/prebuilt (2)
	$ export ARCH=arm
	$ export CROSS_COMPILE=$PWD/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
下载 X-loader	$ cd $PANDA_WORK
	$ git clone git://git.omapzoom.org/repo/x-loader.git (2)
	$ cd x-loader
	$ git checkout -b omap4_dev origin/omap4_dev
下载 U-boot	$ cd $PANDA_WORK
	$ git clone git://git.omapzoom.org/repo/u-boot.git (2)
	$ cd u-boot
	$ git checkout -b omap4_dev origin/omap4_dev
下载内核	$ cd $PANDA_WORK
	$ git clone https://android.googlesource.com/kernel/omap.git kernel (2)
	$ cd kernel
	$ git checkout -b android-omap-panda-3.0 origin/android-omap-panda-3.0

1. 这在现有文档中被视为 RAS （Android Root 源码）。
2. 上述链接在编写时有效 （2014 年 4 月）。

2.3.2 使用需要的补丁

表 3. 使用需要的补丁 （分步说明）

步骤	说明
在 U-boot 中使用补丁	$ cd u-boot
	$ wget http://android-development-environment.googlecode.com/files/0001-change-bootarges.patch (1)
	$ git apply 0001-change-bootarges.patch
使用内核补丁	$cd $PANDA_WORK
	$ cd kernel
	$ wget http://sola-dolphin-1.net/data/Panda/0001-panda-jb4.2_kernel.patch
	$ git apply 0001-panda-jb4.2_kernel.patch

1. 上述链接在编写时有效 （2014 年 4 月）。

2.3.3 编译源码

表 4. 编译源码 （分步说明）

步骤	说明
编译 X-loader	$ cd x-loader
	$ git checkout -b omap4_dev origin/omap4_dev
	$ make omap4430panda_config
	$ make ift
	$ cp -a MLO $ANDROID_ROOT/device/ti/panda/xloader.bin
编译 U-boot	$ cd u-boot
	$ make omap4430panda_config
	$ make
	$ cp -a u-boot.bin $ANDROID_ROOT/device/ti/panda/bootloader.bin
编译内核	$ cd $PANDA_WORK
	$ cd kernel
	$ make panda_defconfig
	$ make
	$ cp -a arch/arm/boot/zImage $ANDROID_ROOT/device/ti/panda/kernel
编译 Android	$ cd $ANDROID_ROOT
	$ source build/envsetup.sh
	$ lunch aosp_panda-userdebug
	$ make –j4

注： 上述链接在编写时有效 （2014 年 4 月）。

2.3.4 烧写 image 文件到 PandaBoard SD 卡

请参考 $ANDROID_ROOT/device/ti/panda/ 中的 README 文件。

3 Linux 内核空间

该层包含 Linux 设备驱动：hts221.ko、 lps25.ko 和 uv.ko。

它们使用输入子系统，这是一种适用于其他输入设备 （包括鼠标和摇杆）的通用 Linux 框架。通过 Sysfs 虚拟文件系统 （/sys/class/input/）导出到用户空间的数据可以在 /dev/input/input<x> 中找到，每个设备具有唯一的 <x>。驱动通过稳定的 Linux 子系统 I²C 从 / 向传感器接收 / 发送数据。

注： 在编写时， hts221 驱动仅在 OneShot 模式下工作。

3.1 环境概述

为了正确配置 PandaBoard，将湿度、气压和紫外线传感器的适配器连接到 I²C 总线上的 PandaBoard （请参考*第 4.1 章*）

然后，如下文所述对 [KR]/arch/arm/mach-omap2/board-omap4panda.c 中的文件 board-omap4panda.c 使用补丁。

3.1.1 I²C 总线初始化补丁

```
static int __init omap4_panda_i2c_init(void)
{
   omap4_pmic_init("twl6030", &omap4_panda_twldata);
   omap_register_i2c_bus(2, 400, NULL, 0);
   /*
    * 将总线 3 连接到设备 （例如 pico DLP）的 DVI 端口
    * 投影仪在频率为 400kHz 时无法可靠工作
    */
   omap_register_i2c_bus(3, 100, panda_i2c_eeprom, ARRAY_SIZE(panda_i2c_eeprom));
   omap_register_i2c_bus(4, 200, panda_i2c_memsensors,
ARRAY_SIZE(panda_i2c_memsensors));
   return 0;
}
```

添加以上以粗体显示的代码行，以便初始化总线 4 （设备所连接的总线）。

还需添加下列结构。

3.1.2 I²C_board_info 结构补丁

```
static struct i2c_board_info __inidata panda_i2c_memsensors[] = {
  {
    I2C_BOARD_INFO("lps25h", 0x5d),
    .platform_data = & lps25h_platform,
  },
  {
    I2C_BOARD_INFO("hts221", 0x5f),
    .platform_data = & hts221_platform,
  },
  {
    I2C_BOARD_INFO("uvis25", 0x47),
    .platform_data = & uvis25_platform,
  }
}
```

3.1.3 platform_data 示例：

```
static struct hts221_platform_data hts221_platform = {
   .poll_interval = 1000,
   .min_interval = 100,
}
```

文件 hts221.h 中有同名结构，用于编译相应设备的模块驱动，位于 [KR]/include/linux/input 中。

3.1.4 重建内核之后

复制合适文件夹 [RAS]/device/ti/panda/ 中的新建 zImage 作为 ‘kernel’，然后重建 boot.img。

- [KR]$ cp -a arch/arm/boot/zImage $ANDROID_ROOT/device/ti/panda/kernel
- [RAS]$ make bootimage

3.2 驱动描述

设备驱动是与硬件的第一个接口；它们通过 i2c 总线直接与传感器通信。每个驱动有两个文件：.c 和 .h 文件。

3.2.1 如何编译和安装设备驱动

为了编译合适的设备驱动， .c 文件必须位于 <KR>/drivers/misc 文件夹，而相应的 .h 文件必须位于 <KR>/include/lilnux/input 中。

如果将驱动作为独立模块进行编译，下面的示例演示了要添加到 <KR>/drivers/misc 下的 Makefile 文件的代码行：

- obj-m = hts221.o
- obj-m + = lps25.o

- obj-m + = uv.ko

然后，从 <KR> 运行指令：“make modules”。

编译模块完成后运行下面的指令：

- [RAS]$ adb root
- [RAS]$ adb remount

按照常用流程将编译出的 hts221.ko、 lps25.ko 和 uv.ko 模块 （位于 <KR>/drivers/misc 下）置于 Android 文件系统中：

- [RAS]$ adb push [modulename].ko /system/lib/hw

然后，可以安装模块并实现功能：

- [RAS]$ adb shell

[Android shell]$ insmod /system/lib/hw/[modulename].ko

3.2.2 从 Linux 用户空间控制设备驱动

可以从 Linux 用户空间控制设备，方法是使用合适的 ‘echo’ 指令从 shell 将所需设置写入相关控制文件，或写入库或应用程序。

上述文件位于 “/sys/class/input/input[x]/device/<busnum>/<i2c-address>/” 目录下，其中 busnum 是总线编号 （在使用 PandaBoard 的本测试示例中为 ‘4’）， i2c 地址随使用的设备而异：

hts221 : 004f

lps25h : 005d

uvis25 : 0047

可以在驱动的相应 .c 文件中找到这些控制文件的名称，其使用的结构是 “attributes”。

最重要的文件为 ‘enable_device’。在本例中， Panda-board 测试环境中 lps25h 的完整路径为：

"/sys/class/input/input[x]/device/4/<4-005d/enable_device"

将其配置为 ‘1’ 可设置设备打开，配置为 ‘0’ （零）可设置设备关闭。

为了从 ‘Linux 用户空间 ’ 设置设备的打开或关闭，使用下列指令：

- [Android shell]$ echo 1 > /sys/class/input/input[x]/device/<busnum>/<i2c-address>/enable_device
- [Android shell]$ echo 0 > /sys/class/input/input[x]/device/<busnum>/<i2c-address>/enable_device

或者：

- [Android shell]$ echo 1 > /sys/bus/i2c/devices/<busnum>/<i2c-address>/enable_device
- [Android shell]$ echo 0 > /sys/bus/i2c/devices/<busnum>/<i2c-address>/enable_device

The devices can be controlled through <sysfs_interface>:

- /sys/bus/i2c/devices/<busnum>/<i2c-address>/

还可以设置其他功能的打开或关闭。

例如对于 lps25h，可以修改 FIFO 设置。

FIFO 模式为：

表 5. FIFO 设置

ID	意义
1	FIFO
2	Stream
6	Mean
3	Stream2FIFO
4	Bypass2Stream
7	Bypass2FIFO

在结构 “attributes” 中，有 “enable_fifo”、 “fifo_mode” 和 “num_samples_fifo”。

在下面的例子中，我们将 fifo_mode 设置为 6、采样数设置为 8：

- [Android shell]$ echo 1 > /<sysfs_interface>/enable_fifo
- [Android shell]$ echo 6 > /<sysfs_interface>/fifo_mode
- [Android shell]$ echo 8 > /<sysfs_interface>num_samples_fifo

另一个例子是，如果现在考虑 hts221，我们可以按以下方式选择加热器或 odr：

打开加热器：

[Android shell]$ echo 1 > /<sysfs_interface>/heater

关闭加热器：

[Android shell]$ echo 0 > /<sysfs_interface>/heater

设置 ODR：

[Android shell]$ echo 'n' > /<sysfs_interface>/poll_period_ms

其中 ‘n’ 为：

1000: 1 hz,

143<'n'<999: 7 hz,

80<'n'<142: 12.5 hz.

设置 OneShot 模式：

[Android shell]$ echo 1 > /<sysfs_interface>/oneshot

3.3 许可设置

ramdisk.img 中某些文件的许可应特别设置，方法是向相应 init.<board>.rc 文件添加某些代码行 （对于 Panda-board，文件 init.omap4pandabpard.rc 位于 <RAS>/device/ti/panda 中）。

尤其是：

chmod 0666 /sys/class/input/input<x>/device/device/enable_device

chown system system /sys/class/input/input<x>/device/device/enable_device

chmod 0666 /sys/class/input/input<x>/device/device/full_scale

chown system system /sys/class/input/input<x>/device/device/full_scale

chmod 0666 /sys/class/input/input<x>/device/device/poll_period_ms

chown system system /sys/class/input/input<x>/device/device/poll_period_ms

其中， <x> 对应于相应事件的 ‘ 编号 ’。

在修改上述文件后，从 <RAS> 运行 “make bootimage” 以编译新的 boot.img。这一新 image 包含内核 （<KR>/arch/arm/boot 中的 zImage）和 ramdisk.img，可以在 <RAS>/device/ti/panda 中找到。

3.4 驱动的输出数据

3.4.1 数据位置

Linux 基础设施从已分配 /dev/input/event<x> 设备的驱动提供原始数据 （参见常规 <device>_report_values，其中 <device> 为设备名称 hts221 或 lps25h 或 uvis）。

可以从 Linux shell 使用相应指令访问此数据。

- [Android shell]$ getevent /dev/input/event<x>

还可以通过 Minicom 读取实际数据 （从设备读取或计算），但只是出于调试目的。为此，可在相应驱动中启用 #DEBUG，然后通过 hts221_get_data() 或 lps25_prs_get_presstemp_data() 中的 pr_info 读取数据。

3.4.2 用于读取数据的应用样例

对于输出到 /dev/event/input<X> 的数据，还可以使用简单的 C 语言算法、动态库或直接从应用程序读取。

下面是一个简单的框架，其中 argv[1] 为路径：/dev/input/event

```
#include <stdio.h>
#include <time.h>
#include <sys/times.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <assert.h>
```

```
struct input_event {
    struct timeval time;
    uint16_t type;
    uint16_t code;
    int32_t value;
};

int main(int argc, char *argv[])
{
    int fd;
    struct input_event ev;
    assert(16 == sizeof(struct input_event));
    if (argc != 2) {
        fprintf(stderr, "missing /dev/input/XXX\n");
        return 1;
    }
    if ((fd = open(argv[1], O_RDONLY)) == -1) {
            perror("open");
            return 1;
    }

    while (1) {
     do
     {
      read(fd, &ev, sizeof(struct input_event));
      if (ev.type == EV_ABS)
        printf("type:%u code:%u value:%d\n", ev.type, ev.code, ev.value);
        usleep(500000);  /* 2 reads per sec */
    } while (ev.type != EV_SYN);
   }

   close(fd);
   return 0;
}
```

当 getevent 指令启动时，类型、时间、代码和值对应的数字会输出到 shell。

在我们的示例中，可以看到气压 （abs_pr）、温度 （选为 abs_gas）和湿度 （选为 abs_misc）的输入值分别为：18、 09 和 28。

4 Android 传感器 HAL

4.1 概述

Android 传感器 HAL 是提供从内核空间驱动程序到 Android 传感器服务和 Android SensorManager 的链路的库。

4.1.1 传感器库

这些库是动态的；它们从 /dev/input/event 中获取数据并提供给上层。通过 SensorManager 类、传感器服务类和传感器 HAL 完成此任务。

Android 框架的 ServiceManager 检查路径 /system/lib/hw 以便了解下列动态库是否存在：

- sensors.default.so
- sensors.<TARGET_BOARD_PLATFORM>.so

4.2 文件

当前考虑的环境传感器为：

- HTS221：相对湿度 + 温度
- LPS25：气压 + 温度
- UVIS25：紫外线

在这一阶段，路径和名称为硬编码；具体而言，即 “lps25h”、 “hts221” 和 “uvis25”，如基础内核驱动所提供。路径可通过 libsensor 找到。

库以使用面向对象方法的 C++ 语言编写。每个传感器都有一个自定义类文件：HumSensor.cpp、 PressSensor.cpp 和 UVSensor.cpp，它扩展了公共基类（SensorBase.cpp）。

4.3 如何编译和安装 Android 传感器 HAL

为了在正确环境中从源码文件开始编译 libsensors.so，必须按照下列说明编译软件包并将其添加到实际传感器 HAL 库：

将传感器 HAL 压缩文件包复制到相关 Android 源码路径，通常位于：

- [Root Android Sources]/device/[vendor name]/[boardname]/

解压缩 “tar -xzvf libsensors_env.tar.gz” （有可能是 .zip 格式）

在编译库之前，初始化 Android 环境：

- [RAS]$ source build/envsetup.sh
- [RAS]$ lunch [target board]

编译库。

在 HAL 文件夹中：

- [RAS]/device/[vendor name]/[boardname]/libsensors

在 HAL 文件夹中启动 “mm” 指令，以便构建名为 sensors.[板名称].so 的动态库。在进程结束时，可以在下列目录中找到库：

- [RAS]/out/target/product/[boardname]/system/lib/hw/

然后，可以将此库添加到现有库，重新挂载文件系统并使用 “adb push” 指令。执行以下步骤：

- [RAS]$ adb root
- [RAS]$ adb remount
- [RAS]$ adb push sensors.[boardname].so /system/lib/hw
- [RAS]$ adb shell] stop
- [RAS]$ adb shell] start

注： *为防止与其他类似的库同名，使用有含义的不同名称，例如 sensors.[处理器名].so。*

就 Panda-board 而言，例如：sensors.omap4.so。

所构建的库名称可以被选择和修改，将所需名称写入：

相关 Android.mk 文件中的 LOCAL_MODULE := sensors.$(TARGET_BOARD_PLATFORM)。

5 为测试构建简单 apk

为了快速构建大致的测试应用，使用常用构建工具 （ADT/Eclipse 等）从 “made-by-default”apk 开始，并修改下列内容：

步骤 1： /res/layout/activity_main.xml

Android apk 样例：activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >


<TextView
   android:id="@+id/TextView01"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Humidity Info">

</TextView>

</LinearLayout>
```

步骤 2：/src/com.example.com.MainActivity.java

```
package com.example.<ProjectName>
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.ViewGroup.LayoutParams;

import java.io.IOException;
import java.io.InputStream;

import android.annotation.SuppressLint;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

@SuppressLint("NewApi")
```

```
public class MainActivity extends Activity implements SensorEventListener{

    private SensorManager mSensorManager;
    private Sensor mHumidity;
    TextView xViewP = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);


        xViewP = (TextView)findViewById(R.id.TextView01);
        xViewP.setText(" Humidity: ");

       // 传感器
       mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
       mHumidity = mSensorManager.getDefaultSensor(Sensor.TYPE_RELATIVE_HUMIDITY);
}

   @Override
   protected void onResume() {
   if(mSensorManager != null)
    mSensorManager.registerListener(this, mHumidity, SensorManager.SENSOR_DELAY_NORMAL);

    super.onResume();
    }

    @Override
   protected void onPause() {
       super.onPause();
    mSensorManager.unregisterListener(this);
    }

   public void onSensorChanged(SensorEvent event) {
      if (event.sensor.getType() == Sensor.TYPE_RELATIVE_HUMIDITY) {
      xViewP.setText("Humidity: " + event.values[0] + "- Temperature: " + event.value[1] );
      }
   }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // 填充菜单；这样可以将项目添加到操作栏 （如果有）中。
```

```
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

以上只是测试环境的简单示例；在构建合适的 apk 时应考虑其他特性。

请注意，将在被测环境传感器中设置的传感器类型为：

LPS25H : Pressure

HTS221：Relative_Humidity

UVIS25: Light

在实际的 Android 应用中，无法预知是否为紫外线类传感器；因此将选项 “Light” 视为正确选择。

6 故障排除

下面是潜在问题列表和相应的解决方案：

- 硬件环境设置：
  - 在使用 shell 指令 “insmod /system/lib/hw/<devicename>.ko” 加载模块时，主机的 “minicom”shell 应显示关于正确探测到设备的消息。
  - 然后，应可以使用路径 /sys/class/input/input[x]/device/<busnum>/<i2c-address> 下的所有文件 （特别是 enable_device）。
- 可通过 minicom 查看数据，但是没有导出到 /dev/input/event<X>
  - input_allocate_device 中可能存在问题；检查整个环境。
  - 另外，确认文件的合适权限。特别是文件 enable_device 和 pollrate_ms 应该至少为 664。
- 数据被输出到 /dev/input/event<X>，但是不能通过 apk 查看。
  - 检查 Linux 用户空间或 Android 环境中的问题。
  - *第 3.4.2 节* 中的应用样例可用于确认数据是否到达 Linux 用户空间。如果是，问题出在 Android 特有的环境中；从主机上的 RAS 尝试下列指令：

    [RAS]$ adb logcat

    或

    [RAS]$ adb logcat | grep sensor

    选择与该主题相关的消息。
  - 更好的选择可能是在文件夹中添加某些特定的日志，以便说明问题的位置。可通过向原始代码中添加 “ALOGI （“ 消息类型 ”） ” 来实现这一点。
- Android Root 源码的 Java 编译问题。
  - 为防止编译在几分钟后失败，确保 Java PATH 和选择设置正确：假设 jdk 位于 /usr/java 中，则：$ PATH=/usr/java/jdk1.6.0_45:$PATH

注： *某些情况下，在路径末尾添加 “/” （即 /usr/java/jdk1.6.0_45/）可能产生编译问题。*

  - $ sudo update-alternatives -- 安装 "/usr/bin/java" "java" "/usr/java/jdk1.6.0_45/bin/java" 1
  - $ sudo update-alternatives -- 安装 "/usr/bin/javac" "javac" "/usr/javac/jdk1.6.0_45/bin/javac" 1
  - $ sudo update-alternatives -- 安装 "/usr/bin/javaws" "javac" "/usr/javaws/jdk1.6.0_45/bin/javaws" 1
  - 在 /etc/environment 中，添加：export JAVA_HOME='usr/java/jdk1.6.0_45/bin
- Linux 正常但是 Android 故障，并且在屏幕上无法查看。
  - 检查与帧缓冲失败相关的错误消息。如果有，按照*第 2.3.1 节*确认下载的 “Imgtec” 图库完美匹配内核中的 PVR 驱动 （在 gpu/pvr/ 中）。

7 关键字

7.1 术语和缩略语

下面是本文档中使用的术语、缩写和缩略语 （按字母顺序排序）。

- ACK - 回应
- APK - Android 应用程序包
- FIFO - 先进先出
- HAL - 硬件抽象层
- HW - 硬件
- JNI - Java 本地接口
- KR - 内核 Root
- PCB - 印刷电路板
- RAS - Android Root 源码：~/panda_work/android
- SoC - 片上系统
- SW - 软件
- TS - 时间戳
- UML - 统一建模语言

8 版本历史

表 6. 文档版本历史

日期	版本	变更
2014 年 6 月 23 日	1	初始版本
2014 年 7 月 29 日	2	更新了封面的标题
2016 年 1 月 21 日	3	更新：*图 2* 和 *第 2.1 节第 6 页*

表 7. 中文文档版本历史

日期	版本	变更
2016 年 3 月 18 日	1	中文初始版本

重要通知 - 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。