sensor 培训课程

课程名称： sensor培训课程

课程类别： 客户培训课程

课程目标：

1. Sensor 分类
2. Sensor porting
3. Sensor layer

对象：客户　　课时数：2H

讲师：驱动工程师　　教学法：面授

Agenda

1. 简介(Introduce)
2. 硬件接口(HW Interface)
3. 软件接口(SW Layer)
4. 代码移植(Porting)
5. Q/A

目前市面上产品主要支持五类传感器：

Acc：重力类传感器。如方向控制、屏幕旋转、翻转控制等。

Mag：磁力类传感器，与Acc配合，生成角度值。如指南针等。

Orientation：方向类传感器，由重力和磁力数据合成。

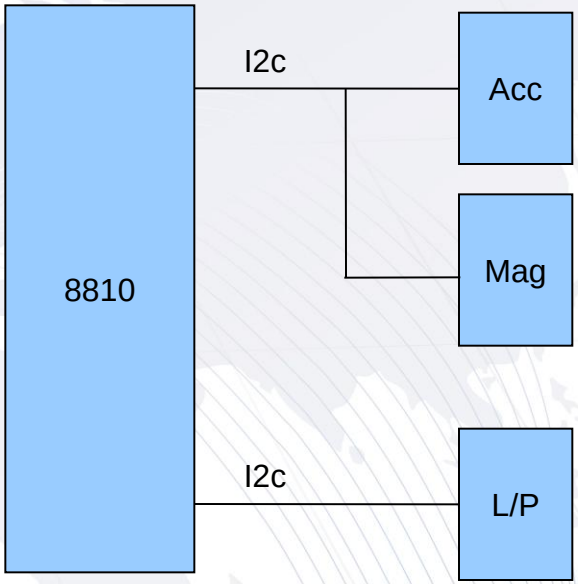
Light：亮度类传感器。应用于亮度控制。

Proximity：接近类传感器。应用于屏幕控制。

在以上传感器中,orientation是需要同时有Acc和Mag数据的，并需要通过算法库才能合成，且以Mag为主。所以Mag厂家就形成了各自的sensor Hal框架。如akm、memsic、freescale等。在项目初期挑选Mag器件时，我们是需要询问下厂家是否有支持的。light、proximity与acc类似，只需要在libsensors中添加相应的hal层接口即可使用。

此文档重点以Mag(akm8975)描述为主，其他的sensor可以类推，控制方式也简单。

HW Interface



- I2C通讯需要2根线(SCL,SDA)。因为其简单、便捷性，已被广泛应用在外围设备中。sensor也是采用此类接口。
- 通过I2C接口，可以对设备进行操作：
  - 器件ID初始化
  - 器件的打开和关闭
  - 获取设备的状态和数据

SW Layer

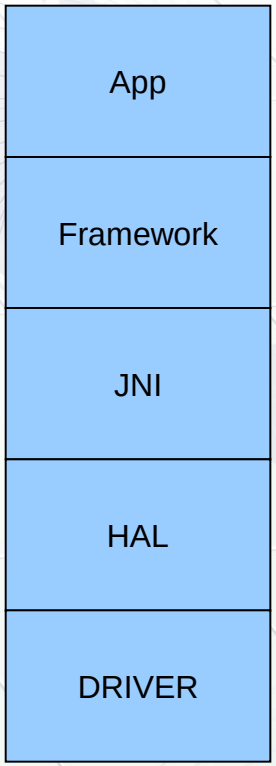
- 每个layer分层涉及的主要文件如下：
- App

  Sensor_data.apk
- Framework
  - SensorManager.java
- Jni
  - SensorService.cpp　　状态控制

  android_hardware_SensorManager.cpp　数据控制
- Hal
  - Libsensors/　　包含sensor hal层代码
  - Sensors.cpp　　hal 接口实现
  - akmd　　Mag服务
- Driver

  -lis3dh_acc.c　　Acc

  -akm8975.c　　Mag



SW Layer – Framework

- 主要函数描述：
- SensorManager.java
- // 使能设备

  -static jboolean

  -sensors_enable_sensor(JNIEnv *env, jclass clazz, jint nativeQueue, jstring name, jint sensor, jint delay)

  -// 获取设备数据

  - static native int sensors_data_poll(int queue, float[] values, int[] status, long[] timestamp);

SW Layer – Jni

主要函数描述:

SensorService.cpp 状态控制

```
// 设备使能
status_t SensorService::SensorEventConnection::enableDisable(int handle,
    bool enabled)
```

- // 设置延时

```
status_t SensorService::SensorEventConnection::setEventRate(int
    handle, nsecs_t ns)
```

android_hardware_SensorManager.cpp 数据控制

```
// 数据获取
static jint sensors_data_poll(JNIEnv *env, jclass clazz, jint
nativeQueue, jfloatArray values, jintArray status, jlongArray
timestamp)
```

SW Layer - Hal

- Sensor hal层的代码最终会生成sensor.sprdbp.so，供系统加载.
- 为了支持sensor应用，在hal层中一般会包含：
  - Akmsensor.cpp：　Mag相关的hal实现
    - int AkmSensor::readEvents(sensors_event_t* data, int count)
    - int AkmSensor::setDelay(int32_t handle, int64_t ns)
    - int AkmSensor::setEnable(int32_t handle, int enabled)
  - lis3dhsensor.cpp ：　Acc相关的hal实现
  - Lightsensor.cpp、Proximitysensor.cpp：　light/prox相关的hal实现
  - Sensorbase.cpp：　Sensor基类实现，所有sensor应该由此继承
  - Sensors.cpp：　Sensor的hal层抽象实现，其中主要函数：
    - int activate(int handle, int enabled);　// 使能
    - int setDelay(int handle, int64_t ns);// 上报频率
    - int pollEvents(sensors_event_t* data, int count);　// 数据包
  - Akmd：Mag&Acc 服务进程，用于读取Acc&Mag原始数据，并通过运算得到准确的Acc/Mag/Orientation数据。
  - 另外，Gyro等其他新增加的hal代码也在此目录添加。

SW Layer – driver

- lis3dh_acc.c　　// acc 驱动实现代码，内容重叠，后续有详细描述
- akm8975.c　　// mag 驱动实现代码
- // 主要的函数和结构
  - module_init(akm8975_init)　　// 模块加载的入口函数
  - module_exit(akm8975_exit)　　// 模块加载的出口函数
  - static struct i2c_driver akm8975_driver = {}　　// I2C设备驱动程序
  - // 设备检测函数，在成功注册i2c设备驱动后，I2C总线会通过.prob函数查找设备是否存在
  - int akm8975_probe(struct i2c_client *client, const struct i2c_device_id *id)
  - // 在卸载模块时，清理注册的设备信息
  - static int akm8975_remove(struct i2c_client *client)

  - // 这块涉及到动态加载和静态加载问题，其实两种方式工作方式一样，只是向I2C总线注册的时机不同
  - #define I2C_BUS_NUM_STATIC_ALLOC
  - #define I2C_STATIC_BUS_NUM　　(0)
  - static struct i2c_board_info　akm8975_i2c_boardinfo = {

    I2C_BOARD_INFO(AKM8975_I2C_NAME, AKM8975_I2C_ADDR),

    }

  -

Porting

- 因为sensor的hal层有了sensors.h的抽象，所以sensor的porting过程实现上就是hal和driver的porting。
- 下面会先以先driver，后hal的方式讲解。

Porting-driver-akm8975.c

- // 在设备采用insmod加载时，入口就是module_init
- module_init(akm8975_init);
- static int __init akm8975_init(void)
- {
  - int ret = -1;
  - printk(KERN_INFO "AKM8975 compass driver: initialize.");
  #ifdef I2C_BUS_NUM_STATIC_ALLOC
  - ret = i2c_static_add_device(&akm8975_i2c_boardinfo);
  - if (ret < 0) {
  - pr_err("%s: add i2c device error %d\n", __FUNCTION__, ret);
  - return ret;
  - }
  -#endif
  - return i2c_add_driver(&akm8975_driver);
- }
- 此处涉及一个概念：静态加载和动态加载。
- 此份代码采用的是静态加载。当akm8975_init函数被调用时，首先程序向i2c总线注册了akm8975设备信息，包含设备名和设备地址，信息包含在akm8975_i2c_boardinfo中，以便通知总线此设备需要侦测；然后向i2c总线注册了akm8975驱动程序，告诉i2c总线可以通过.prob函数进行设备侦测。2个结构体如下：

Porting-driver-akm8975.c

- static struct i2c_board_info akm8975_i2c_boardinfo = {
  I2C_BOARD_INFO(AKM8975_I2C_NAME, AKM8975_I2C_ADDR),
  };
- AKM8975_I2C_NAME 向i2c总线注册的设备名
- AKM8975_I2C_ADDR 设备的i2c读写地址

- static struct i2c_driver akm8975_driver = {
  .probe   = akm8975_probe,
  .remove   = akm8975_remove,
  .id_table = akm8975_id,
  .driver = {
  .name = AKM8975_I2C_NAME,
  },
  };
- akm8975_probe：函数监测设备是否真实存在。
  akm8975_remove：在设备被卸载时，此函数清除probe注册的信息
  AKM8975_I2C_NAME：2个结构体中的名称要相同，不然i2c总线找不到设备。

Porting-driver-akm8975.c

- int akm8975_probe(struct i2c_client *client, const struct i2c_device_id *id)
- {
  - ...
  - // 注册名为compass的input设备，用于sensor数据的上报。

  err = akm8975_input_init(&s_akm->input);

  - ...

  // 注册了misc设备，通过ioctl方式，获取设备的状态和控制设备的打开、关闭等。

  err =misc_register(&akm8975_dev);

  - ...

  // 注册了sysfs接口，通过sysfs接口，获取设备的状态和控制设备的打开、关闭等。

  err =create_sysfs_interfaces(s_akm);

  ...
- }
- 函数还包含一些错误处理函数，irq中断配置等没有列出，可自行查看原始代码。
- 在上面的接口中，input接口因为传递的是设备实时的数据，所以都会注册。misc设备和sysfs设备选择一种即可，没有固定的模式。在hal层代码适配时，根据drive提供的接口编写就行。

Porting-driver-akm8975.c

- static int akm8975_input_init(struct input_dev **input)

```
-{
-    *input = input_allocate_device();
-    if (!*input)
-        return -ENOMEM;

-    /* Set name */
-    (*input)->name = "compass";

-    /* Register */
-    err = input_register_device(*input);
-}
```

此处设定了input设备名称为compass，这在akmsensors.cpp中也有体现。但在driver层代码中只是创建了compass设备，并没有对其进行写入操作。写入操作应该在akmd的服务进程中进行，这部份在后续会描述。

Porting-driver-akm8975.c


```
// misc 设备注册时，需要几个结构体，定义如下：
static int AKECS_Open(struct inode *inode, struct file *file);
static int AKECS_Release(struct inode *inode, struct file *file);
static long AKECS_ioctl(struct file *file,
        unsigned int cmd, unsigned long arg);

static struct file_operations AKECS_fops = {
  .owner = THIS_MODULE,
  .open = AKECS_Open,
  .release = AKECS_Release,
  .unlocked_ioctl = AKECS_ioctl,
};

static struct miscdevice akm8975_dev = {
  .minor = MISC_DYNAMIC_MINOR,
  .name = "akm8975_dev",
  .fops = &AKECS_fops,
};
```

- akm8975_dev: misc设备注册成功后，会在/dev/目录下会生成设备文件akm8975_dev，后续就可以通过此设备名操作设备
- AKECS_ioctl: misc设备的分类控制接口

Porting-driver-akm8975.c


```
static long
AKECS_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    void __user *argp = (void __user *)arg;
    struct akm8975_data *akm = file->private_data;
    …
    switch (cmd) {
     case ECS_IOCTL_GETDATA:
     AKM_DATA(&akm->i2c->dev, "IOCTL_GETDATA called.");
     ret = AKECS_GetData(akm, sensor_buf, SENSOR_DATA_SIZE);
     if (ret < 0)
          return ret;
     break;
     case ECS_IOCTL_SET_YPR:
          AKM_DATA(&akm->i2c->dev, "IOCTL_SET_YPR called.");
          AKECS_SetYPR(akm, ypr_buf);
          Break;
    }
    ...
}
```

Porting-driver-akm8975.c

```
static int create_sysfs_interfaces(struct akm8975_data *akm)
{
  // 注册compass class类
  akm->compass = class_create(THIS_MODULE, compass_class_name);
  if (IS_ERR(akm->compass)) {

                        err = PTR_ERR(akm->compass);
                goto exit_class_create_failed;
  }
  …
        // 通过compass类，在/dev目录创建名为akm8975的设备
  akm->class_dev = device_create(akm->compass,NULL,akm8975_device_dev_t,akm,akm8975_device_name);
  …
        // 创建sysfs 链接文件
  err = sysfs_create_link(&akm->class_dev->kobj,&akm->i2c->dev.kobj,device_link_name);
         …
  // 创建akm8975_attributes包含的属性文件，如enable_mag、enable_acc等
  err = create_device_attributes(akm->class_dev,akm8975_attributes);
  …
  // 创建akm8975_bin_attributes包含的二进制文件
  err = create_device_binary_attributes(&akm->class_dev->kobj,akm8975_bin_attributes);
}
```

此函数实现的sysfs文件接口较全面，实际使用时是可以精简的。

Porting-driver-akm8975.c


```
static struct device_attribute akm8975_attributes[] = {
__ATTR(enable_acc, 0660, akm8975_enable_acc_show, akm8975_enable_acc_store),
__ATTR(enable_mag, 0660, akm8975_enable_mag_show, akm8975_enable_mag_store),
__ATTR(enable_ori, 0660, akm8975_enable_ori_show, akm8975_enable_ori_store),
__ATTR(delay_acc,  0660, akm8975_delay_acc_show,  akm8975_delay_acc_store),
__ATTR(delay_mag,  0660, akm8975_delay_mag_show,  akm8975_delay_mag_store),
__ATTR(delay_ori,  0660, akm8975_delay_ori_show,  akm8975_delay_ori_store),
#ifdef AKM8975_DEBUG_IF
__ATTR(mode,  0220, NULL, akm8975_mode_store),
__ATTR(bdata, 0440, akm8975_bdata_show, NULL),
__ATTR(asa,   0440, akm8975_asa_show, NULL),
#endif
__ATTR_NULL,
};

static struct bin_attribute akm8975_bin_attributes[] = {
__BIN_ATTR(accel, 0220, 6, NULL,
NULL, akm8975_bin_accel_write),
__BIN_ATTR_NULL
};
```

Porting -hal-akmsensor.cpp

SPREADTRUM
Innovation changes the future

- Hal层的代码位于libsensors目录下,前面已经介绍过主要文件的作用，下面直接描述akmsensor.cpp代码中。

- // Akmsensor继承于SensorBase，并对相应的成员进行重写。此函数打开名为compass的input设备，也就是akm8975驱动代码中创建的input设备，打开后就可以监控input上的信息变化，获取设备数据。
- AkmSensor::AkmSensor(): SensorBase(NULL, "compass"), mPendingMask(0), InputReader(32){
  - ...
  - // 添加Acc、Mag、Orientation的配置信息
  - memset(mPendingEvents, 0, sizeof(mPendingEvents));
  - mPendingEvents[Accelerometer].version = sizeof(sensors_event_t);
  - mPendingEvents[Accelerometer].sensor = ID_A;
  - mPendingEvents[Accelerometer].type = SENSOR_TYPE_ACCELEROMETER;
  - mPendingEvents[Accelerometer].acceleration.status = SENSOR_STATUS_ACCURACY_HIGH;
  - mPendingEvents[MagneticField].version = sizeof(sensors_event_t);
  - mPendingEvents[MagneticField].sensor = ID_M;
  - mPendingEvents[MagneticField].type = SENSOR_TYPE_MAGNETIC_FIELD;
  - mPendingEvents[MagneticField].magnetic.status = SENSOR_STATUS_ACCURACY_HIGH;
  - mPendingEvents[Orientation].version = sizeof(sensors_event_t);
  - mPendingEvents[Orientation].sensor = ID_O;
  - mPendingEvents[Orientation].type = SENSOR_TYPE_ORIENTATION;
  - mPendingEvents[Orientation].orientation.status = SENSOR_STATUS_ACCURACY_HIGH;
  - ...
  - // 增加sysfs 文件操作路径
  - strcpy(input_sysfs_path, "/sys/class/compass/akm8975/");
- }

Porting -hal-akmsensor.cpp

```
// 设备使能
int AkmSensor::setEnable(int32_t handle, int enabled){
  // 获取sysfs操作的文件名称，即是akm8975.c中创建的sysfs属性
  strcpy(&input_sysfs_path[input_sysfs_path_len], "enable_acc");
  ...
  // sysfs写操作
  err = write_sys_attribute(input_sysfs_path, buffer, 1);
        ...
}
// 延时设置
int AkmSensor::setDelay(int32_t handle, int64_t ns) // 与setEnable操作方式相同
// 数据获取和上传，此处读取的是akm8975.c中创建的compass的input设备。其他的sensor的处理方式相同。
int AkmSensor::readEvents(sensors_event_t* data, int count)
{
  ...
    while (count && mInputReader.readEvent(&event)) {
        if (type == EV_ABS) {
                        processEvent(event->code, event->value);          // 分类存储sensor的数据
                              mInputReader.next();
        } else if (type == EV_SYN) {
              *data++ = mPendingEvents[j];                          // 通过data 将数据上报到JNI
    }
}
```

Porting -hal-sensors.cpp

Sensors.cpp文件为hal接口的抽象实现。

```
// 定义驱动层上使用的sensor参数，jni层可以根据此数组了解设备列表
static const struct sensor_t sSensorList[]
static int sensors__get_sensors_list(struct sensors_module_t* module,
                                     struct sensor_t const** list)

// 添加sensor的监控句柄
sensors_poll_context_t::sensors_poll_context_t();

// 定以设备可以使用的功能
static int open_sensors(const struct hw_module_t* module, const char* id,
                        struct hw_device_t** device)
{
    ...
            dev->device.common.close    = poll__close;
    dev->device.activate        = poll__activate;
    dev->device.setDelay        = poll__setDelay;
    dev->device.poll            = poll__poll;
    ...
}
```

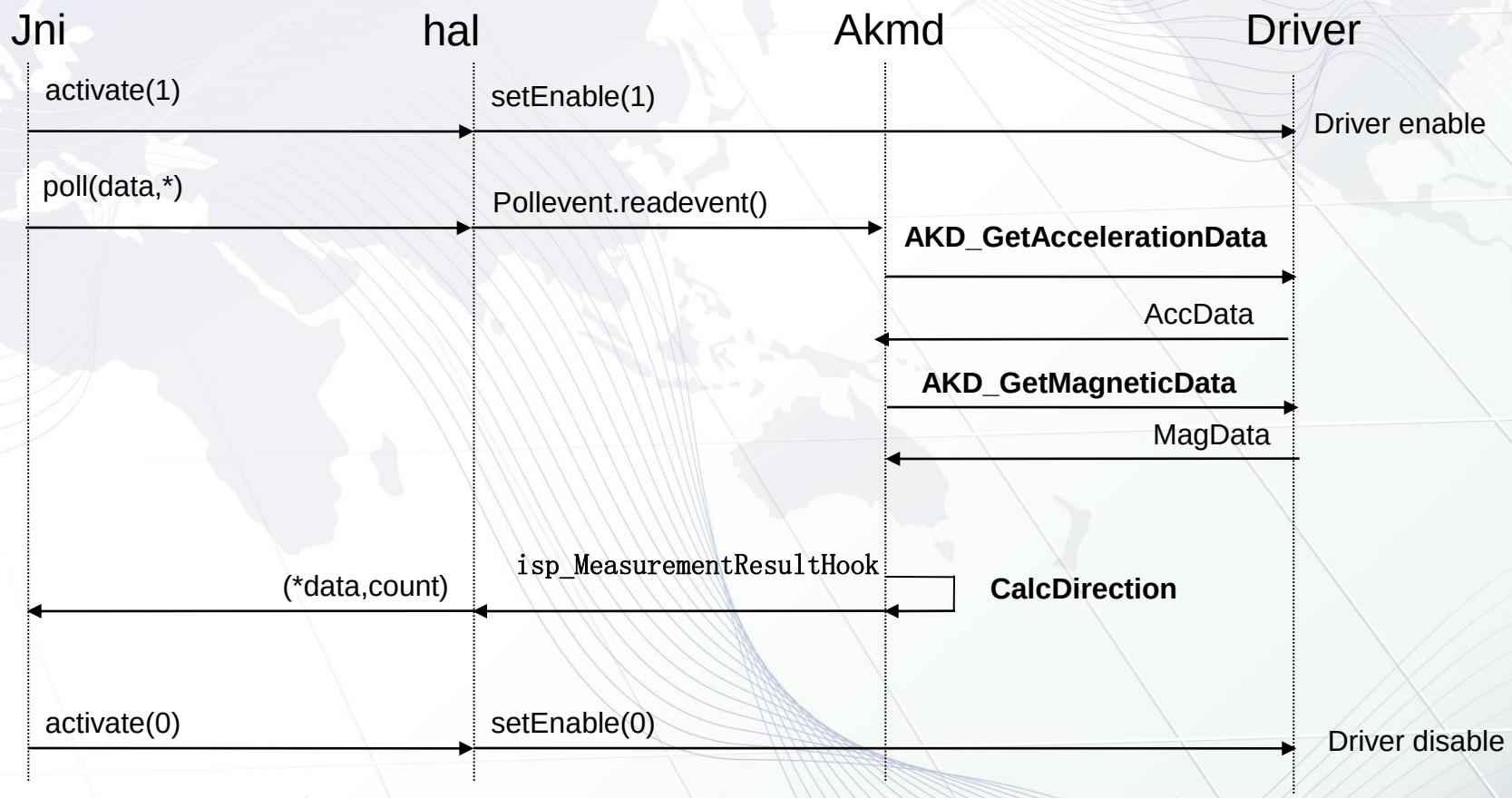
Porting -hal-sensors.cpp

SPREADTRUM
Innovation changes the future

```
// Hall 重要的结构体
struct sensors_module_t HAL_MODULE_INFO_SYM = {
    common: {
        tag: HARDWARE_MODULE_TAG,
        version_major: 2,
        version_minor: 0,
        id: SENSORS_HARDWARE_MODULE_ID,
        name: "SPRD Sensors Module",
        author: "SPRD Inc.",
        methods: &__module_methods,
        reserved: {0}
    },
    get_sensors_list: __get_sensors_list
};
```

Porting -hal-akmd

整个移植过程还缺少个akmd的服务进程,服务的工作方式如下：



在设备上报数据时，采用input方式上报。而设备使能等操作，就需要hal层和driver统一采用一种方式。Mag数据因为需要进行orientation换算，所以流程中添加了akmd服务。所以可以看到，在mag驱动中，没有直接向input设备更新数据，只是在ioctl接口中封装了供akmd使用。Acc, light, proximity等设备不需要akmd服务进程，可以直接在代码使用input_report_abs()和input_sync()，将数据上报到hal。

Porting -hal-akmd

涉及的主要函数如下：

1. Android.mk.3rdparty

   akmd服务进行的编译脚本

2. main.c

```
int main(int argc, char **argv)
{
  while (g_mainQuit == AKD_FALSE)
   {
            ...
            /* Read Parameters from file. */
            if (LoadParameters(&prms) == 0) {
            SetDefaultPRMS(&prms);
            }
            ...
            /* Start measurement thread. */
            if (startClone(&prms) == 0)
            ...
            if (AKD_GetCloseStatus(&st) != AKD_SUCCESS)
            ...
   }
}
```

Porting -hal-akmd

SPREADTRUM
Innovation changes the future

```
// start new thread
      static int startClone(AK8975PRMS* prms)
{
      pthread_attr_t attr;
      pthread_attr_init(&attr);
      g_stopRequest = 0;
      if (pthread_create(&s_thread, &attr, thread_main, prms) == 0)
      {
               return 1;
      }
      else
      {
               return 0;
      }
}
static void* thread_main(void* args)
{
      MeasureSNGLoop((AK8975PRMS *) args);
      return ((void*)0);
}
```

Porting -hal-akmd

3. measure.c

```
// This is the main routine of measurement.
void MeasureSNGLoop(AK8975PRMS* prms)
{
  if (exec_flags & (1 << (ACC_MES_FLAG_POS))) {
        AKD_GetMagneticData(i2cData);
        GetMagneticVector()
  }
  if (exec_flags & (1 << (ACC_MES_FLAG_POS))) { AKD_GetAccelerationData(); }
        if (exec_flags & (1 << (ORI_ACQ_FLAG_POS))) {
        CalcDirection();
        }
  }
  if (exec_flags & 0x0F) {
        Disp_MeasurementResultHook();
  }
}
// Output measurement result
void Disp_MeasurementResultHook(AK8975PRMS* prms, const uint16 flag)
{
        AKD_SetYPR(rbuf);     // ECS_IOCTL_SET_YPR
}
```

Porting -hal-akmd

4. misc.c

```
int openInputDevice(const char* name)// get input devices
// get realy point data  因为各厂家的贴片方式不一样，此函数转化sensor的数据为lib库需要的数据，便于计算。
int16 ConvertCoordinate(const AKMD_PATNO    pat, int16vec*vec)
```

5. AK8975Driver.c

```
// Open device driver.
int16_t AKD_InitDevice(void)
// Writes data to a register of the AK8975.
int16_t AKD_TxData()   {
        if (ioctl(s_fdDev, ECS_IOCTL_WRITE, buf) < 0)
}

// Acquires data from a register or the EEPROM of the AK8975.
int16_t AKD_RxData(const BYTE address,BYTE * data,const uint16_t numberOfBytesToRead)
{
        ...
        if (ioctl(s_fdDev, ECS_IOCTL_READ, buf) < 0)
        ...
}
```

Porting -hal-akmd

```
//Acquire magnetic data from AK8975.

int16_t AKD_GetMagneticData(BYTE data[SENSOR_DATA_SIZE])

void AKD_SetYPR(const int buf[YPR_DATA_SIZE])
int AKD_GetOpenStatus(int* status)
int AKD_GetCloseStatus(int* status)

int16_t AKD_SetMode(const BYTE mode)

int16_t AKD_GetDelay(int64_t delay[AKM_NUM_SENSORS])

//  贴片方向

int16_t AKD_GetLayout(int16_t* layout)

//Acc data

int16_t AKD_GetAccelerationData(int16_t data[3])
```

Porting 一些说明

- 到此为止，akm8975的driver和hal代码的移植过程完成。
- 在此文档中还存在一些知识点没有描述。如misc，input，sysfs等。如有兴趣，可以研究下。

Porting-acc

SPREADTRUM
Innovation changes the future

Lis3dhSensor.cpp // acc 适配层，不同厂家文件名称不同，工作原理类似

```
#define LIS3DH_ACC_DEV_PATH_NAME    "/dev/lis3dh_acc"
#define LIS3DH_ACC_INPUT_NAME  "accelerometer"
Lis3dhSensor::Lis3dhSensor()
: SensorBase(LIS3DH_ACC_DEV_PATH_NAME, LIS3DH_ACC_INPUT_NAME),...);
// LIS3DH_ACC_DEV_PATH_NAME, LIS3DH_ACC_INPUT_NAME 对应于lis3dh_acc.c中probe中注册的dev和input设备名称
int Lis3dhSensor::setEnable(int32_t handle, int enabled);
int Lis3dhSensor::setDelay(int32_t handle, int64_t delay_ns);
int Lis3dhSensor::readEvents(sensors_event_t* data, int count);
```

lis3dh_acc.c // acc driver ， 不同的厂家名称可能不同

```
#define  LIS3DH_ACC_DEV_NAME  "lis3dh_acc"
//acc->input_dev->name = "accelerometer";

static int lis3dh_acc_probe(struct i2c_client *client,
  const struct i2c_device_id *id)
{
  err = lis3dh_acc_input_init(acc);
  err = misc_register(&lis3dh_acc_misc_device);
}
```

在调试driver时，如果出现设备不能打开时，可能对照相应的设备probe函数中注册的设备，使用adb shell中查找相应的设备是否正常建立。

Porting-light&proximity

SensorAL3006.cpp

```
SensorAL3006::SensorAL3006()
  : SensorBase(AL3006_DEVICE_NAME, "proximity"),...
{
    memset(mPendingEvents, 0, sizeof(mPendingEvents));
    mPendingEvents[Light].version = sizeof(sensors_event_t);
    mPendingEvents[Light].sensor = ID_L;
    mPendingEvents[Light].type = SENSOR_TYPE_LIGHT;
    mPendingEvents[Proximity].version = sizeof(sensors_event_t);
    mPendingEvents[Proximity].sensor = ID_P;
    mPendingEvents[Proximity].type = SENSOR_TYPE_PROXIMITY;
}
```

AL3006 的hal适配层驱动是合在一起的，所以在构造函数时，捕捉的事件需要添加2个，如上。当然，在驱动设备层的代码，light和proximity是可以分为2个分别处理的。

```
int SensorAL3006::setDelay(int32_t handle, int64_t ns)
int SensorAL3006::setEnable(int32_t handle, int en)
int SensorAL3006::getEnable(int32_t handle)
int SensorAL3006::readEvents(sensors_event_t* data, int count)
```

Porting-light&proximity

SPREADTRUM
Innovation changes the future

al3006_pls.c

```
static int al3006_pls_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
  //register device
  err = misc_register(&al3006_pls_device);
  input_dev->name = AL3006_PLS_INPUT_DEV;
  err = input_register_device(input_dev);
}
```

1. 设备不能加载？

   1）insmod不能加载时，根据kmsg提示的信息，逐步排查错误，直到probe函数能正常执行完成。

   2）可以分别在xxx_init()、xxx_probe中增加调试信息

2. 改动的hal层代码不起作用？

   在主编译脚本中作了些处理，在搜索到目录\idh.code\customize\customer_cfg\sp8810ga\proprietar中存在sensors.sprdbp.so时，直接使用此so,而不是编译新的。在修改libsensor目录时，需要注意下。

3. i2c not ack

   1）注意i2c的地址是否正确，i2c地址应为实际的7bit IC 地址

   2）注意检查I2C总线配置是否正确，8810上有3个i2c总线。

www.spreadtrum.com

谢谢