UDS710_UDX710 Android 10.0

SensorHub客制化指导手册

WWW.UNISOC.COM
紫 光 展 锐 科 技

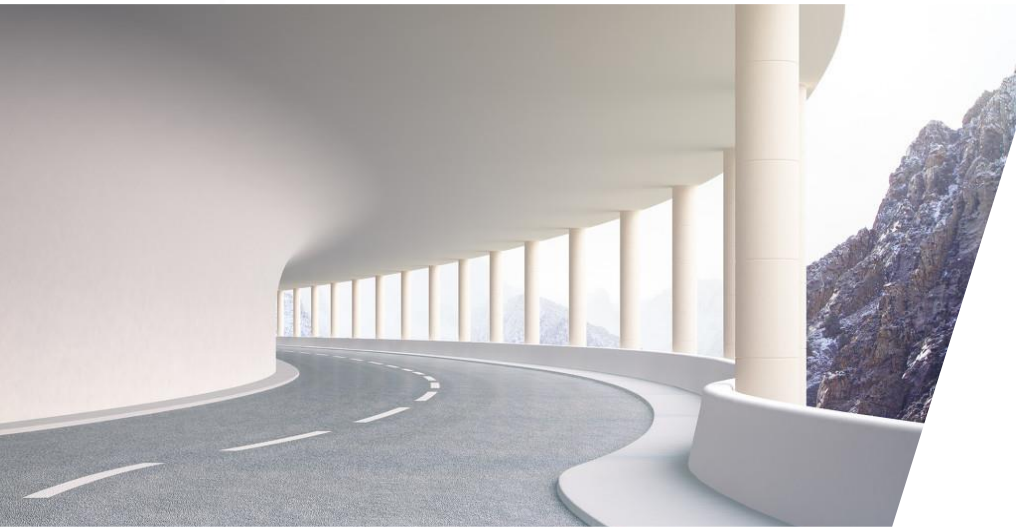
修改历史

版本号	日期	注释
V1.0	2020/10/10	第一次正式发布。

关键字

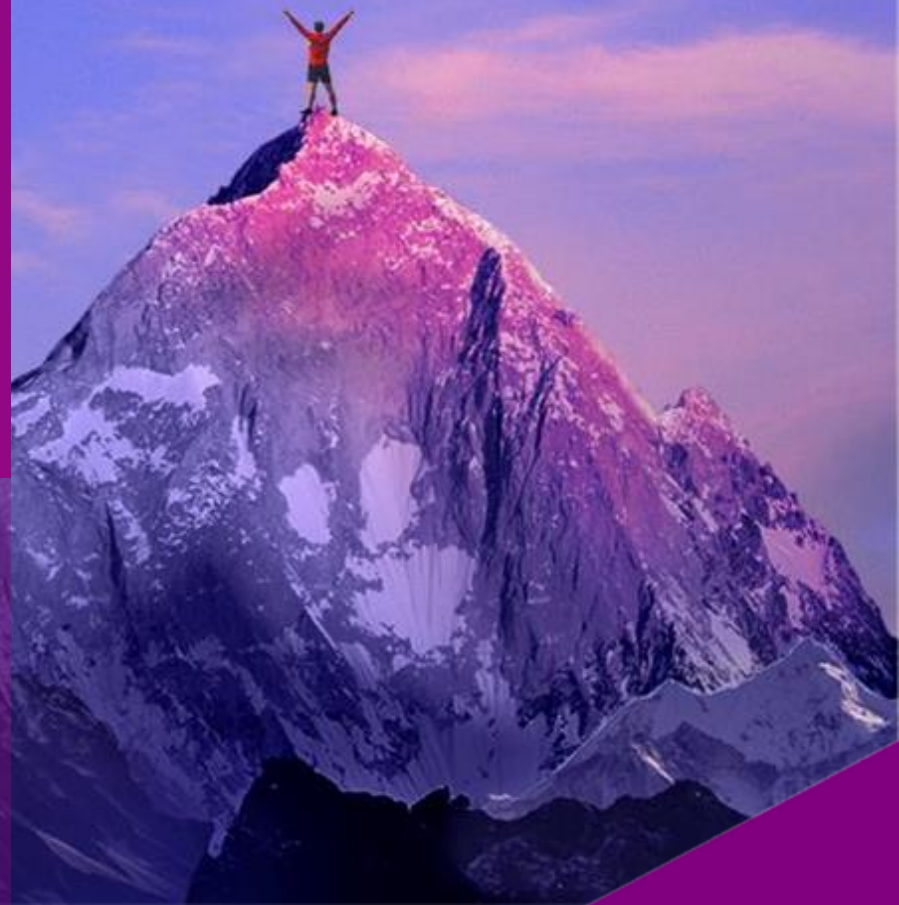
关键字： SensorHub、客制化

目录



01

SensorHub 代码结构介绍



SensorHub 代码结构简介

AP

SensorHub HAL层代码

路径： vendor/sprd/modules/sensors/libsensorhub

HAL层实现Android定义的标准Sensors HAL接口、Sensor特性参数配置、接口定义配置。

SensorHub Kernel层代码

路径： bsp/kernel/kernel4.14/drivers/iio/sprd_hub

Kernel层代码主要负责将HAL层下发命令及Sensor配置参数传递给CM4，将CM4反馈的信息及上报的Sensor事件传递给HAL层。

SP

SensorHub CM4代码

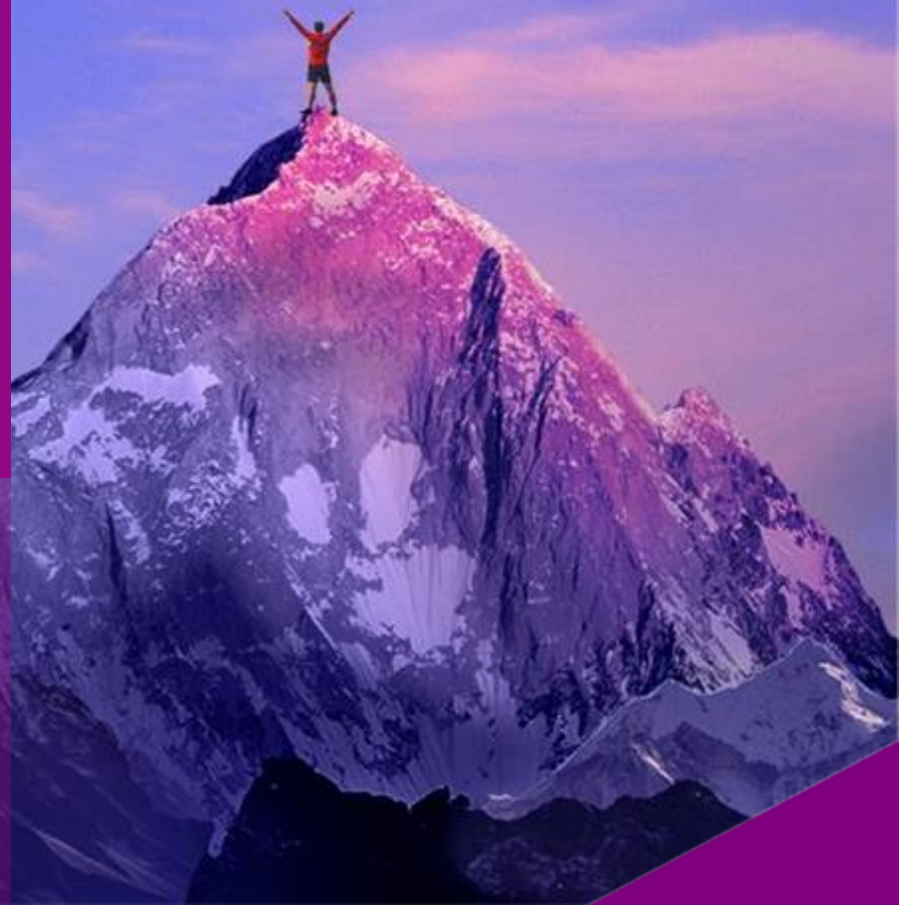
Sensor驱动架构及SensorHub算法放置在CM4侧。目前代码闭源，仅提供bin文件供客户直接打包使用（随modem版本一起发布）。
bin文件为roc1_cm4.bin。

CM4 动态加载代码

对于需要特殊操作的Sensor，CM4提供动态加载方式，由客户自行编写动态加载代码，例如地磁算法库就是用动态加载方式实现。
UDS710_UDX710动态加载代码在IRAM的执行位置为0x3B000，SensorHub提供给动态加载的接口固定位置为0x3AFA0

2

SensorHub 客制化步骤



SensorHub 客制化步骤 (1/2)

1. 配置Sensor型号及类型

配置Sensor型号

路径：device/sprd/roc1/ud710_2h10/BoardConfig.mk

示例：

```
SENSOR_HUB_ACCELEROMETER := lsm6dsl_ud710
SENSOR_HUB_GYROSCOPE := lsm6dsl
SENSOR_HUB_LIGHT := ltr578als ltr553als
SENSOR_HUB_MAGNETIC := akm09918_ ud710
SENSOR_HUB_PROXIMITY := ltr578als ltr553als
SENSOR_HUB_PRESSURE := null
```

说明：

赋值的名字需与opcode文件名中的（Sensor型号）一致。赋值null表示不支持此类型Sensor。同类型Sensor，如需兼容多个型号，型号以空格隔开，如上面LIGHT所示，同时兼容ltr578als和ltr553als。

配置Sensor类型

路径：device/sprd/roc1/ud710_2h10/ud710_2h10_Base.mk

示例：

```
PRODUCT_COPY_FILES += \
frameworks/native/data/etc/android.hardware.sensor.accelerometer.
xml:$(TARGET_COPY_OUT_VENDOR)/etc/permissions/android
.hardware.sensor.accelerometer.xml \
frameworks/native/data/etc/android.hardware.sensor.compass.xml:$
(TARGET_COPY_OUT_VENDOR)/etc/permissions/android.hard
ware.sensor.compass.xml \
frameworks/native/data/etc/android.hardware.sensor.gyroscope.xml
:$(TARGET_COPY_OUT_VENDOR)/etc/permissions/android.har
dware.sensor.gyroscope.xml \
```

说明：根据需要配置的Sensor类型添加相应的代码行。

SensorHub 客制化步骤 (2/2)

2. 增加对应Sensor参数配置文件(opcode)

在以下路径中增加(Sensor类型)_(Sensor型号).cpp文件。
路径：vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/(Sensor类型)/

示例：
以ud710_2h10工程添加加速计lsm6dsl为例，增加文件：
vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/accelerometer/accelerometer_lsm6dsl_ud710.cpp

说明：(Sensor型号)需与device路径下的工程配置中的Sensor类型赋值一致。

3. 注册opcode接口

路径：vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/

示例：
以加速度计lsm6dsl为例。

- 在SensorHubOpCodeExtrator.h中增加以下代码：
  ```
  extern void SensorHubOpcodeRegisterAccelerometer_lsm6dsl_ud710();
  ```

- 在SensorHubOpCodeExtrator.cpp中增加以下代码：
  ```
  #ifdef SENSORHUB_WITH_ACCELEROMETER_lsm6dsl_ud710
      strcat(accelerometer_list, "accelerometer_lsm6dsl_ud710,");
      SensorHubOpcodeRegisterAccelerometer_lsm6dsl_ud710();
  #endif
  ```

03
Sensor
特性参数
opcode
配置说明

Sensor 特性参数opcode配置说明

- Sensor 功能寄存器配置
- Sensor 特性信息配置
- Sensor 校准参数配置

Sensor 功能寄存器配置 （1/4）

在Sensor特性参数配置(opcode)文件中，需将Sensor初始化、使能、关闭、采样速率等寄存器配置到对应功能配置数组。

iic_unit原型

```
typedef struct iic_unit {
    uint8_t operate;
    uint8_t addr;
    uint8_t val;
    uint8_t mask;
} iic_unit_t;
```

Sensor 功能寄存器配置（2/4）

iic_unit成员取值说明

成员	取值说明
operate	0x01：Read，读操作。
	0x02：Write，写操作。
	0x03：Check，检测操作，比如检测某一位是否置1。
	0x04：Delay，延时操作，单位为ms。
	0x05：操作控制CM4侧GPIO高低电平。
	0xff：在getrawdata配置中，如果Sensor i2c不支持连续读寄存器，则在最后增加一行operate值为0xff的配置。
addr	寄存器地址、GPIO端口号。
val	• operate为写操作时，为需要写入的值。 • operate为检测操作时，为要检测对比的值。 • GPIO电平配置。
mask	• operate为delay操作时，延时时间值，要延时多少ms。 • operate为write、read、check操作时，需要处理的bit位置1，其余都置0。

Sensor 功能寄存器配置（3/4）

功能类型说明

Opcode类型	数组名称	定义
CMD_HWINIT	eCmdInitArray	第一次上电，初始化Sensor的配置，只在刚开机的时候执行一次。
CMD_ENABLE	eCmdEnableArray	填写每次enable的时候需要操作的寄存器，每次Enable Sensor的时候都会执行。
CMD_DISABLE	eCmdDisableArray	填写每次disable的时候需要操作的寄存器，每次Disable Sensor的时候都会执行。
CMD_GET_BYPASS	eCmdGetRawDataArray	填写读取raw data的寄存器地址，顺序为从低位到高位，以加速度为例：X_L，X_H，Y_L，Y_H，Z_L，Z_H。
CMD_SET_STATUS	eCmdSetStatusArray	有些Sensor需要写入才能清掉状态位。
CMD_SET_SELFTEST	eCmdSetSelftestArray	填写selftest的相关信息。
CMD_SET_OFFSET	eCmdSetOffsetArray	设置Sensor的offset寄存器，如果使用Sensor自身的校准功能会用到，但目前有专门的校准算法，所以此项配置不用填写。
CMD_SET_RATE	eCmdSetRateArray	设置Sensor ODR (Output Data Rate)相应信息，填写顺序为由快到慢，以加速度计为例，数组中依次填入100Hz，50Hz，16Hz，5Hz相应的寄存器。
CMD_SET_MODE	eCmdSetModeArray	设置工作模式。
CMD_GET_STATUS	eCmdGetStatusArray	当读取Sensor raw data前会读Sensor status寄存器，status ok代表当前raw data寄存器可读。
CMD_CHECK_ID	eCmdCheckIdArray	填写Sensor ID寄存器地址及ID值。

Sensor 功能寄存器配置（4/4）

以bmi160 acc enable的enable操作寄存器配置为例进行说明：

```
static struct iic_unit eCmdEnableArray[] = {
        /* 写操作，往0x7E寄存器写入0x11*/
        {
                .operate = 0x02, .addr = 0x7E, .val = 0x11, .mask = 0xFF,
        },
        /* delay操作，delay 20ms*/
        {
                .operate = 0x04, .addr = 0x00, .val = 0x00, .mask = 0x14,
        },
        /* check操作， 检测bit4 & bit5位是否分别为 “1” 和 “0”*/
        {
                .operate = 0x03, .addr = 0x03, .val = 0x10, .mask = 0x30,
        },
        /* 写操作，往0x41寄存器的低四位写入 “1000”，高四位保持不变*/
        {
                .operate = 0x02, .addr = 0x41, .val = 0x08, .mask = 0x0F,
        },
};
```

Sensor 特性信息配置（1/3）

对sensorInfoConfigArray 进行配置。

sensorInfoConfigArray 原型：

```
typedef struct {
  uint8_t Interface;
  uint16_t Interface_Freq;
  uint8_t ISR_Mode;
  uint8_t GPIO_ISR_Num;
  uint8_t Slave_Addr;
  uint8_t Chip_Id;
  float Resolution;
  uint16_t Range;
  uint8_t Postion;
  uint8_t Vendor;
} sensor_info_t;
```

Sensor 特性信息配置（2/3）

sensor_info_t成员说明：

成员	说明
Interface	通信接口。0代表使用SensorHub I2C0连接，1代表使用I2C1。目前仅有2路I2C，不可填写其他值。
Interface_Freq	通信频率，以KHz为单位。比如目前I2C以400KHz的频率去读写，填写400即可，当前固定使用400KHz。
ISR_Mode	是否使用中断模式。目前尚未支持中断模式。
GPIO_ISR_Num	使用中断模式时用到的GPIO号。
Slave_Addr	Sensor 的I2C写地址，即设备地址右移1位。
Chip_Id	Sensor的chip id。
Resolution	Sensor的精度值。从Sensor中读取的raw data拼接之后会直接乘以此参数，得到的就是标准单位的值：acc(m/s2)，gyro(rad/s)，mag(μT)，pressure(hPa)，light(lux)。可以让Sensor的FAE提供此值，也可以根据spec计算得出。
Range	量程。根据Google的标准，ACC一定要配置成 ±8G，ACC的range要填写8，其余的按照spec上的实际配置填写。目前gyro的为2000，mag的为4，其余的不需填写。

Sensor 特性信息配置（3/3）

sensor_info_t成员说明：

<table>
<tr><th>成员</th><th>说明</th></tr>
<tr><td>Position</td><td>调整Sensor数据的方向。Postion值对应的方向调整如下所示：
<table>
<tr><th>Position值</th><th>方向调整</th><th>Position值</th><th>方向调整</th></tr>
<tr><td>0</td><td>X : Y : Z</td><td>4</td><td>Y : X : -Z</td></tr>
<tr><td>1</td><td>Y : -X : Z</td><td>5</td><td>X : -Y : -Z</td></tr>
<tr><td>2</td><td>-X : -Y : Z</td><td>6</td><td>-Y : -X : -Z</td></tr>
<tr><td>3</td><td>-Y : X : Z</td><td>7</td><td>-X : Y : -Z</td></tr>
</table></td></tr>
<tr><td>Vendor</td><td>• 磁力计：此值需与磁力计校准库的mag_init接口返回值一致。
• Prox Sensor：
– 0: 不需将低噪值回写到Sensor，使用展锐内置的快速校准算法。
– 1: 需要回写工厂校准的低噪值到Sensor，使用展锐内置的快速校准算法。
– 0xFE: 使用动态加载机制。
• Acc、Gyro：
– 0： 非动态加载机制。
– 0xFE: 使用动态加载机制。
• Light：
– 0: LTR553。
– 1: Sensor的raw data寄存器读出值为标准照度单位lux，不需要转换。
– 0xFE: 使用动态加载机制。</td></tr>
</table>

说明：对于动态加载机制，请参考《SensorHub动态加载驱动指导》和《EmBitz编译Sensorhub动态加载驱动介绍》。

Sensor 校准参数配置

目前只需配置距感Sensor校准参数，其他Sensor无需配置，下面对距感校准参数取值进行说明。

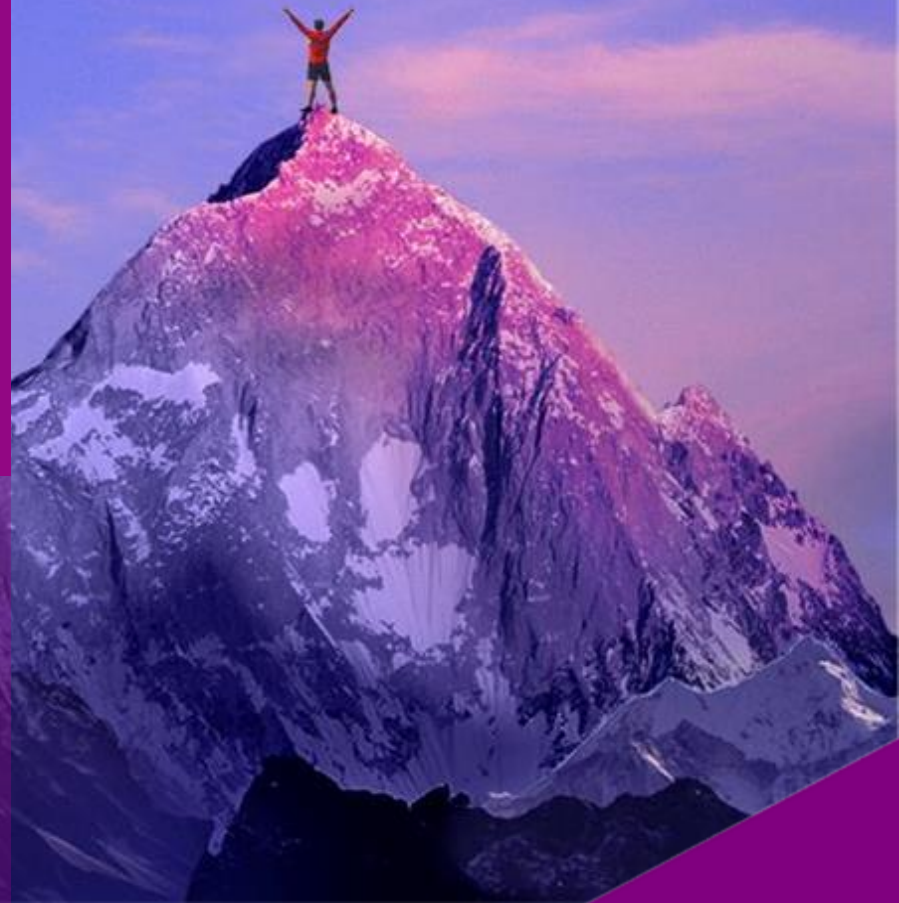
在进行校准参数配置前，建议使用10台左右整机，获取以下数据：

1. 用18%灰卡遮挡距感3cm处，Sensor测量值。
2. 用18%灰卡遮挡距感5cm处，Sensor测量值。
3. 无遮挡时Sensor测量值(底噪)。

读上述值可以将手机root后，打开距感，在shell命令下cat d/sensor/raw_data_ps

```
static prox_cali_t eProxCaliArray[] = {
    {
    .collect_num = 7, //无需修改
    .ps_threshold_high = 0x70, //灰卡3cm时的值，建议使用多台样机中较大的值，无工厂校准时使用
    .ps_threshold_low = 0x30 ,//灰卡5cm时的值，建议使用多台样机中较大的值，无工厂校准时使用
    .dyna_cali = 0xa00, //取sensor满量程值即可，软件自动校准时用于保存最新底噪
    .dyna_cali_add = 0x10, //无遮挡时底噪（raw_data）的浮动值
    .noise_threshold = 0xC0, //灰卡1cm时的值，取多台平均值，用于判断校准时手机是否被遮挡
    .noise_high_add = 0x58, //取ps_threshold_high – 底噪（raw_data）值，用于根据底噪改变阀值
    .noise_low_add = 0x18, // 取ps_threshold_low –底噪（raw_data）值，用于根据底噪改变阀值
    .ps_threshold_high_def = 0x70, //一般与ps_threshold_high取相同值；作为动态校准失效时的ps_threshold_high值
    .ps_threshold_low_def = 0x30, //一般与ps_threshold_low取相同值；作为动态校准失效时的ps_threshold_low值
    .dyna_cali_threahold = 0x18, //取正常样机底噪最小值，底噪值小于此值就不再进行动态校准
    .dyna_cali_reduce = 0x8, //取值和high_add相同，用于过滤异常情况
    .value_threshold = 0xC0, //取值和noise_threshold相同，用于过滤异常情况
    .noise_reference = 0x20, //设置值应为正常sensor底噪值上限，取正常样机无遮挡时噪声最大值，建议与sensor厂商一起确认。此值用于工厂校准时判断
    sensor是否正常，手机底噪大于此值，说明sensor有问题或产线组装异常
    },
};
```

04
SensorHub CM4 log抓取和debug方法



SensorHub CM4 log抓取和debug方法 （1/3）

SensorHub log

- **Log存放位置**
  - SensorHub驱动相关log分3部分：SensorHub所在处理器log，AP侧SensorHub Kernel驱动log，AP侧SensorHub HAL层log。
  - 开机阶段前期（未检测到storage/emulated/0）SensorHub的log保存在data/ylog。
  - 开机阶段后期（检测到storage/emulated/0）SensorHub的log保存在storage/emulated/0/ylog。
  - 可以在ylog配置中设置log开关、改变路径。
  - 进入ylog配置菜单的方法： *#*#83781#*#*->DEBUG&TOOL->Ylog
  - SensorHub处理器log在ylog/sensorhub目录下，AP侧log在ylog/ap文件夹，查看ap log需安装python2.7，运行analyzer.py进行解码。
  - 如果需要在HAL层log中打印出Sensor上报事件详细数据log，在adb shell(root模式)下：echo “4 2” > /d/sensor/logctl。
- **实时读取SensorHub log**

  可以在adb shell下使用如下方法直接查看SensorHub相关实时log：
  - 查看SensorHub HAL层log：logcat | grep SensorHub
  - 查看SensorHub Kernel层log： dmesg –w | grep sensorhub （需要root权限)
  - 查看SensorHub处理器系统log：cat /dev/slog_pm | grep SENSORHUB （需要确保SensorHub log处于打开状态，且手机已经root。如果SensorHub log关闭，可以 echo “log on” > /dev/sctl_pm打开SensorHub处理器系统log。）

SensorHub CM4 log抓取和debug方法（2/3）

常用debug手段

- **查看AP和SensorHub通信是否正常**

  adb shell下(root模式下)，cat /d/sensor/version，通信正常，SensorHub正常工作时，返回一个有效版本值。

  示例：

  ud710_2h10:/ # cat /d/sensor/version
  567
  以上的567即为一个有效版本值。

- **查看Sensor是否初始化成功**

  adb shell下(root模式下)， cat /d/sensor/sensor_info

  示例：

  ud710_2h10:/ # cat /d/sensor/sensor_info
  1 acc name:accelerometer_lsm6dsl_ud710
  1 mag name:magnetic_akm09918_ud710
  1 gyro name:gyroscope_lsm6dsl
  1 prox name:proximity_ltr578als
  1 light name:light_ltr578als
  0 prs name:
  每行开头的数字1代表初始化成功，0代表初始化失败，后面紧跟Sensor类型和Sensor具体型号名称。
  以上情况代表加速计、地磁、陀螺仪、距感、光感均初始化正常，prs(气压计)初始化失败。

SensorHub CM4 log抓取和debug方法（3/3）

- **在线读写Sensor寄存器**
  1. 执行adb root，adb shell，进入d/sensor目录。
  2. 执行cat shub_debug，会有如图1所示输出。
  3. 根据提示执行需要的操作。

  示例：
  - 读加速度计 0x75寄存器：
    执行echo “2 0 0x75 0x00 0xff” > shub_debug，然后cat shub_debug，最后一行输出value值0x11即所读寄存器值，status为0x1表示读取成功，如图2所示。
  - 将0x13写入加速度计0x19寄存器：
    执行echo “3 0 0x19 0x13 0xff” > shub_debug即可。

图1

```
ud710_3h10:/d/sensor # cat shub_debug
cat shub_debug

Description:
        echo "op sid reg val mask" > shub_debug
        cat shub_debug

Detail:
         op(operator): 0:open 1:close 2:read 3:write
         sid(sensorid): 0:acc 1:mag 2:gyro 3:proximity 4:light 5:pressure 6:new_dev
         reg: the operate register that you want
         value: The value to be written or show read value
         mask: The mask of the operation

         status: the state of execution. 1:success 0:fail

[shub_debug]operate:0x0 sensor_id:0x0 reg:0x0 value:0x0 status:0x0
```

图2

```
         value: The value to be written or show read value
         mask: The mask of the operation

         status: the state of execution. 1:success 0:fail

[shub_debug]operate:0x2 sensor_id:0x0 reg:0x75 value:0x11 status:0x1
```

谢 谢