UNISOC Sensorhub
-- for customer

Unisoc

2019年06月

History

Release Date	2019.07.03
Document No.	
Version	V1.0
Document Type	Introduction
Platform	SC9863A/UMS312/UMS512/ UDS710
OS Version	Android8.1/Android9.0

目录

硬件架构

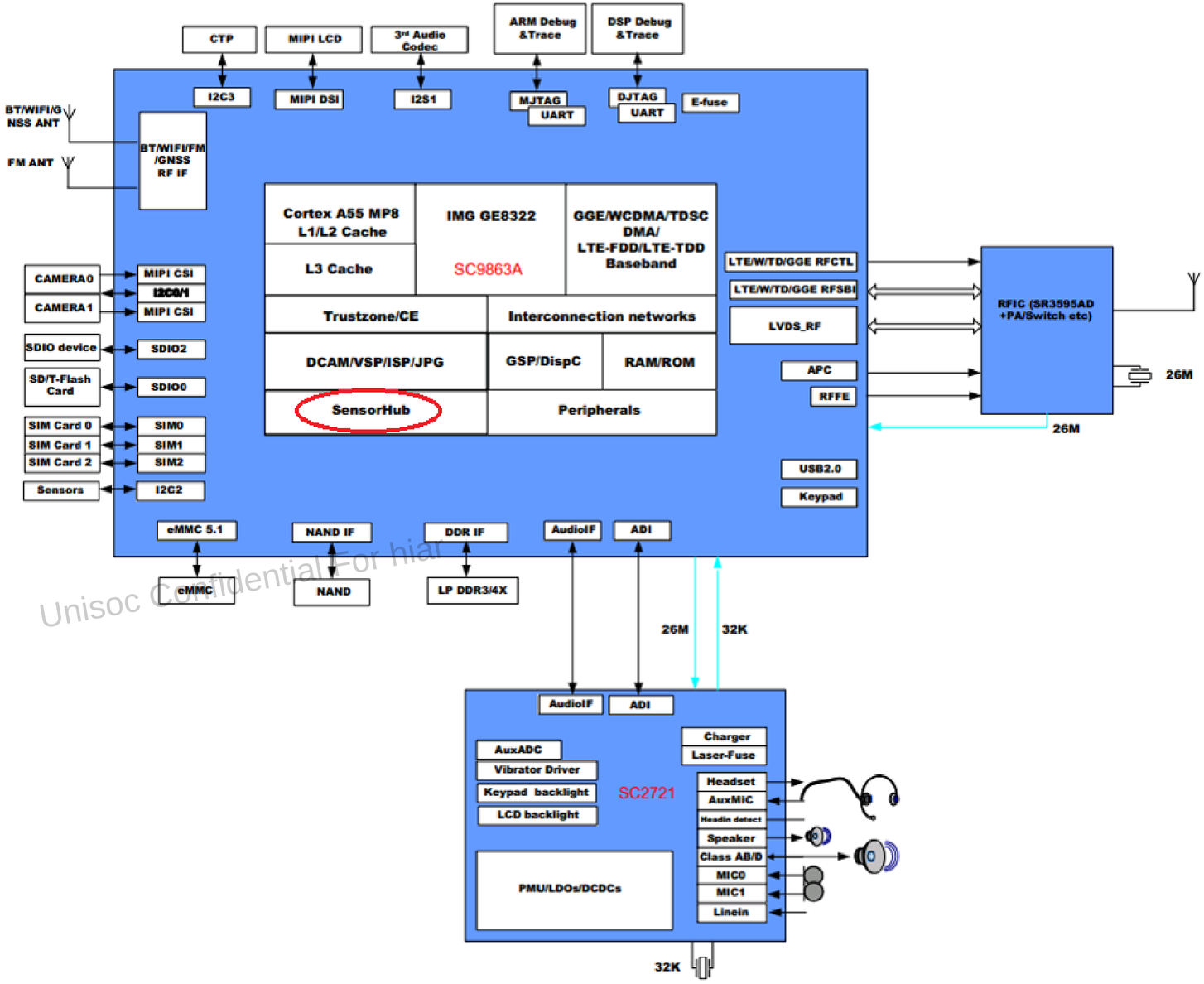
展锐平台 Sensorhub 需要芯片硬件支持。各类不同功能的Sensor IC挂靠在Cortex-M4（下面简称CM4）上，由CM4统一管理。CM4位于CP侧，通过SIPC方式与AP通信。对比Sensor IC挂靠在AP的传统架构，Sensorhub具有功耗低等优势。

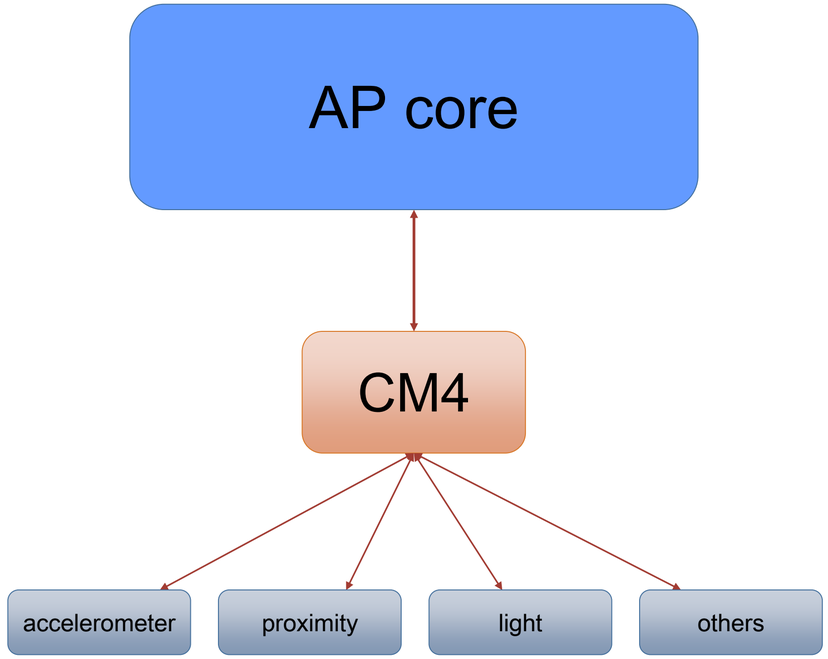
目前支持Sensorhub架构的芯片平台有：

- sc9863a（main）
- ums312
- ums512
- uds710

注：客户量产软件使用sensorhub始于sprdroid7.0（sp9853i），之后在sprdroid8.1（sc9863a）大量使用。下面以sc9863a sprdroid9.0继续介绍。

SC9863A平台Sensorhub简介





目录

软件架构

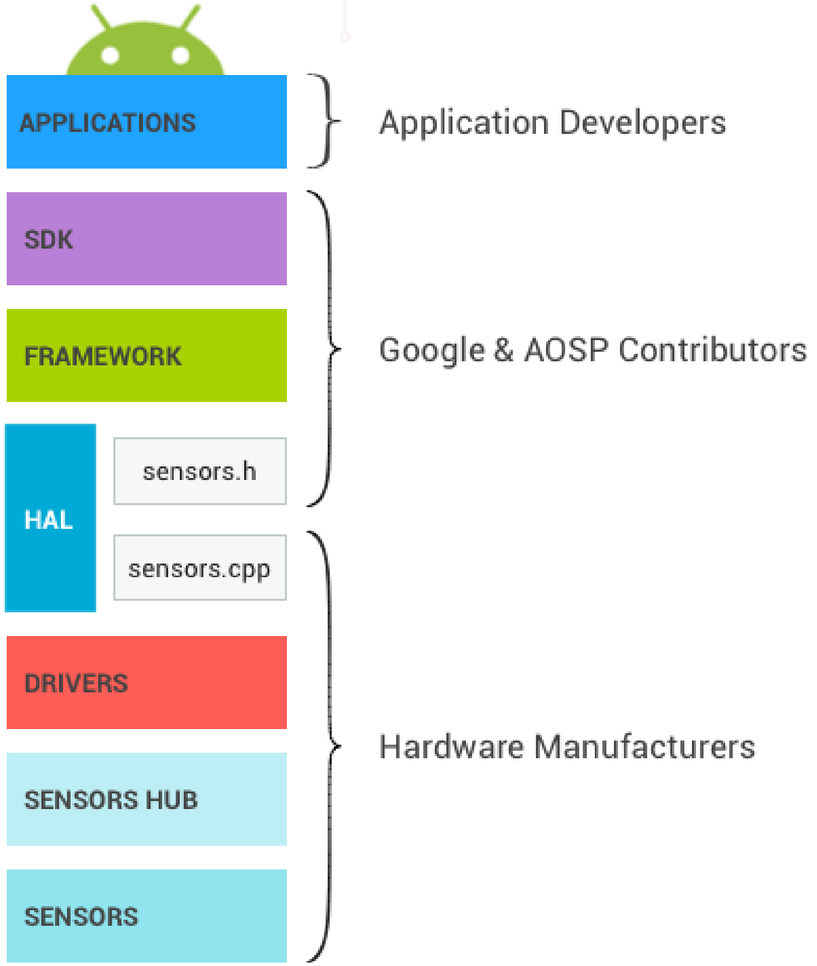
Sensorhub 软件框架

- Overview
- HAL
- Kernel Driver
  - Data protocol
- CM4 Driver
- Dynamic Call

说明：文件目录：

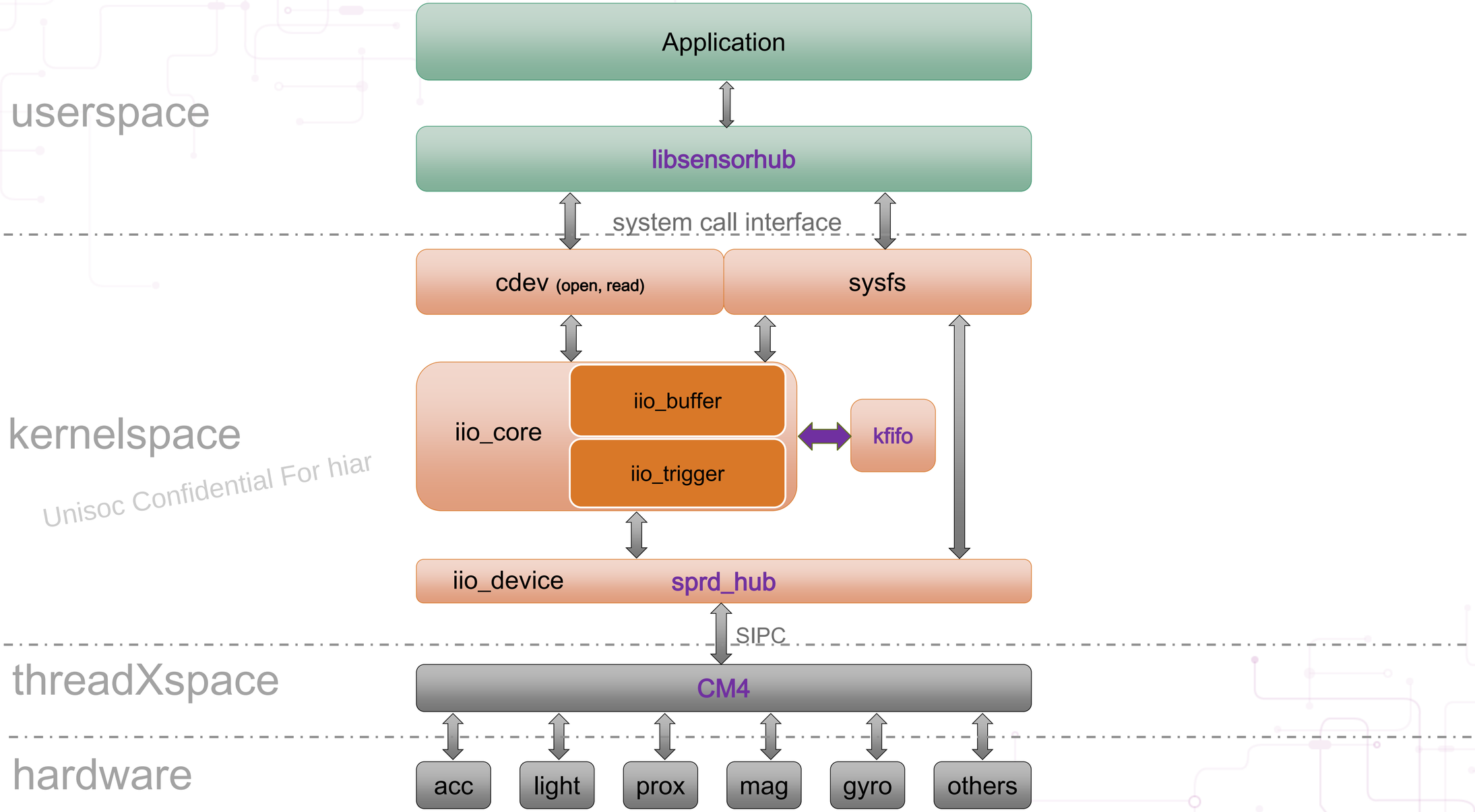
1) HAL
   vendor/sprd/modules/sensors/libsensorhub
2) Kernel Driver
   kernel4.4/drivers/iio/sprd_hub or
   kernel4.14/drivers/iio/sprd_hub
3) CM4 Driver
   Not open for customer
4) Dynamic Call
   patch_table_sensor.c

软件架构--Overview



说明：本文档只关注 Hardware Manufacturers 即平台部分，Application 和 Google 请查阅网络相关资料。

软件架构--Overview



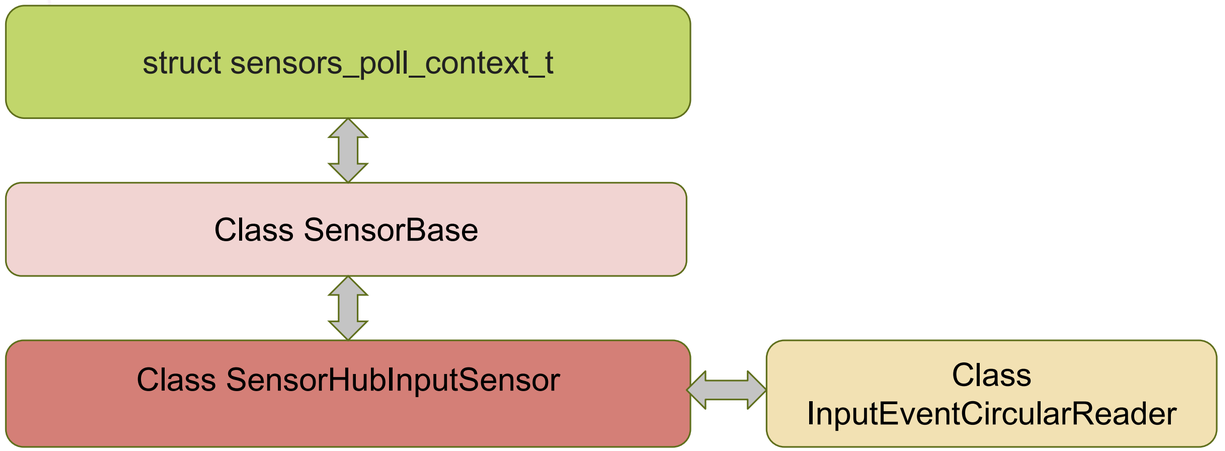
软件架构

Sensorhub 软件框架

- Overview
- HAL
- Kernel Driver
  - Data protocol
- CM4 Driver
- Dynamic Call

软件架构--HAL--Class

Hal层总体框图：



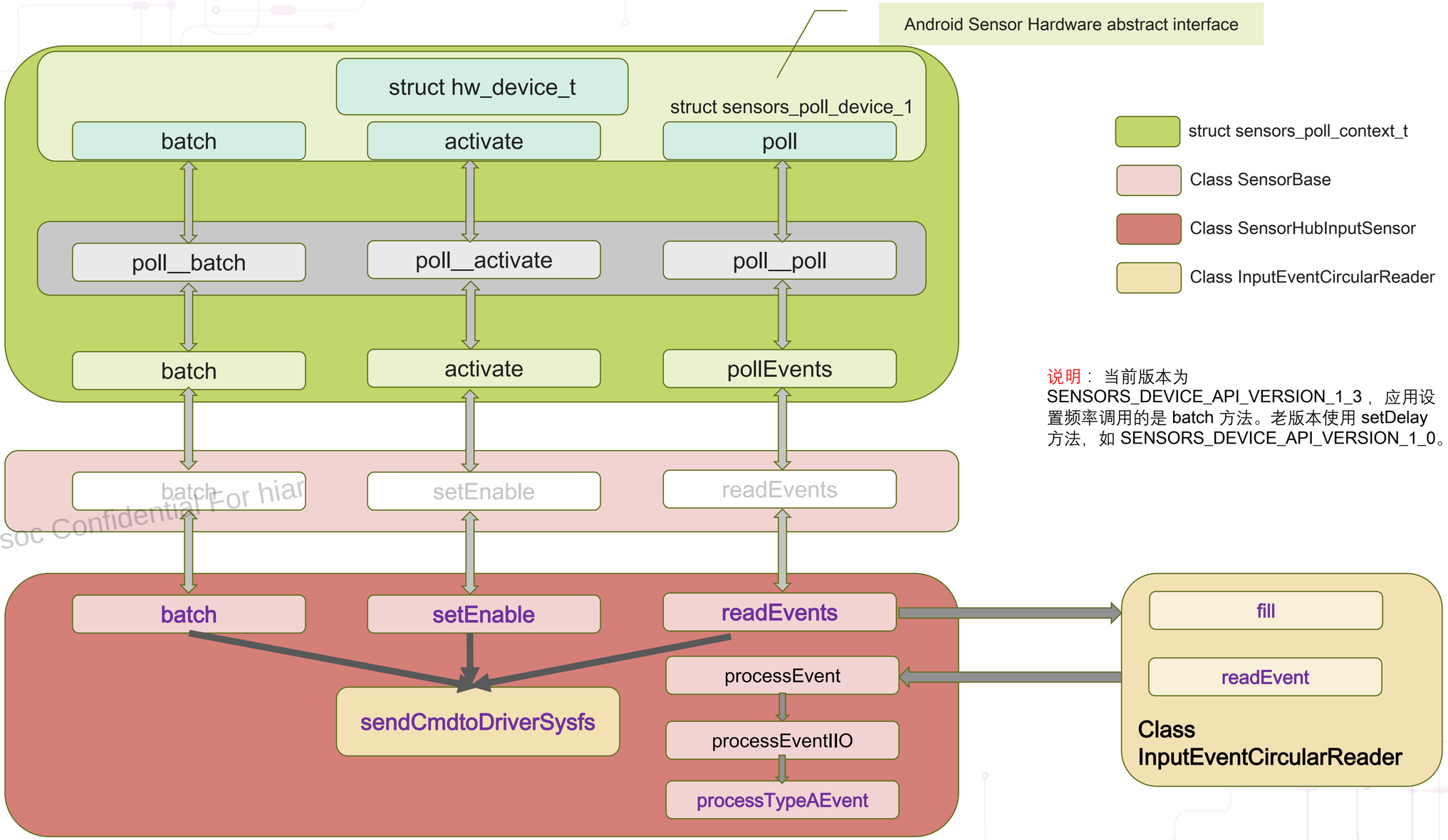
- Struct sensors_poll_context_t：
  Android硬件层sensor抽象接口。

- Class SensorBase：
  平台实现的硬件抽象层基类，抽象类。

- Class SensorHubInputSensor：
  继承于SensorBase的子类，实现对底层驱动的流程控制和数据读取。

- Class InputEvetnCircularReader：
  辅助功能类。处理从底层驱动读来的数据并上传给 APP。

软件架构 -- HAL -- flow



说明：当前版本为 SENSORS_DEVICE_API_VERSION_1_3，应用设置频率调用的是 batch 方法。老版本使用 setDelay 方法，如 SENSORS_DEVICE_API_VERSION_1_0。

软件架构 -- HAL -- open_sensors

open_sensors 是 sensorhub 在HAL层的入口和初始化方法。位于libsensorhub\sensors.cpp。值得注意的是，这里会触发kernel 驱动解析并下载 opcode firmware，sensor info 以及校准参数到 CM4。

```
/** Open a new instance of a sensor device using name */
static int open_sensors(const struct hw_module_t* module, const char* id,
                        struct hw_device_t** device) {
    int status = -EINVAL;
    SH_LOG("id[%s]", id);
    /*
     * Write Opcode first
     * extrate and create sensor firmwares, which include sensor info, opcode, cali array(just prox).
     * */
    SensorHubOpcodeExtrator();

    /* get drv config
     * "new SensorHubInputSensor" for mSensors[] mPollFds[]
     * */
    sensors_poll_context_t *dev = new sensors_poll_context_t();

    memset(&dev->device, 0, sizeof(sensors_poll_device_t));

    dev->device.common.tag      = HARDWARE_DEVICE_TAG;
    dev->device.common.version  = SENSORS_DEVICE_API_VERSION_1_3;
    dev->device.common.module   = const_cast<hw_module_t*>(module);
    dev->device.common.close    = poll__close;
    dev->device.activate        = poll__activate;
    dev->device.setDelay        = poll__setDelay;
    dev->device.poll            = poll__poll;
    dev->device.batch           = poll__batch;
    dev->device.flush           = poll__flush;

    *device = &dev->device.common;
    status = 0;

    return status;
} ? end open_sensors ?
```

软件架构 -- HAL -- methods

1. 控制

int SensorHubInputSensor::downloadFirmware(void)

系统开机SensorHubInputSensor类被创建对象时，在构造函数中被执行，读sys节点“op_download”触发驱动下载opcode到CM4.

int SensorHubInputSensor::sendCmdtoDriverSysfs(int operate, int* para)

通过写对应sys节点来控制底层驱动做相应操作。
节点有“enable”,“delay”,“batch”,“flush”,"sensorhub“.

软件架构 -- HAL -- methods

2. 数据

- 读取

ssize_t InputEventCircularReader::fill(int fd)

从“/dev/iio:device1”中读取数据暂存到HAL层自己维护的环形缓冲区。其中“/dev/iio:device1”是sensorhub kernel driver设备节点。

说明：设备编号也许会变，这由系统iio设备的个数和启动加载的时机而定。

ssize_t InputEventCircularReader::readEvent(pseudo_event const** events)

判断环形缓冲区是否有数据可读，并返回数据地址指针

- 上报

sensors_event_t * SensorHubInputSensor::processEventIIO(pseudo_event const* incomingEvent)

解析处理kernel driver通过iio上报来的数据集合，数据集合被分为以下子类做分类处理：包含eHalCmd， eHalCmd， eHalSenData，eHalFlush，eMcuReady和eCommonDriverReady

sensors_event_t * SensorHubInputSensor::processTypeAEvent(int handle, pmSenMsgTpA_t data)

解析eHalSenData情况下的iio 数据，主要是各类sensor的raw data。

软件架构 -- HAL -- others

1. Non wake up and wake up sensor

- None wake up sensor

非唤醒传感器是不阻止 SoC 进入挂起模式也不会将 SoC 唤醒以报告数据的传感器。

- Wake up sensor

与非唤醒传感器相反，唤醒传感器会确保其数据传输不依赖于 SoC 状态。当 SoC 唤醒时，唤醒传感器的行为与非唤醒传感器相同。当 SoC 休眠时，唤醒传感器必须唤醒 SoC 以发送事件。它们必须仍可让 SoC 进入挂起模式，但是当需要报告事件时，还必须唤醒 SoC。也就是说，在达到最大报告延迟时间或硬件 FIFO 已满之前，传感器必须唤醒 SoC 并传送事件。

- 定义

在 KitKat 及更早版本中，某个传感器是唤醒传感器还是非唤醒传感器取决于传感器类型：除近程传感器和大幅度动作检测器之外，大多数传感器都是非唤醒传感器。

从 L 版本开始，传感器是否为唤醒传感器由传感器定义中的标记指定。大多数传感器均可由同一传感器的唤醒和非唤醒变体对来定义，在这种情况下，它们必须作为两个独立的传感器运行，且彼此不进行交互。

除非在传感器类型定义中另行指定，否则建议为传感器类型中列出的每种传感器类型各实现一个唤醒传感器和一个非唤醒传感器。

参考：https://source.android.com/devices/sensors/suspend-mode

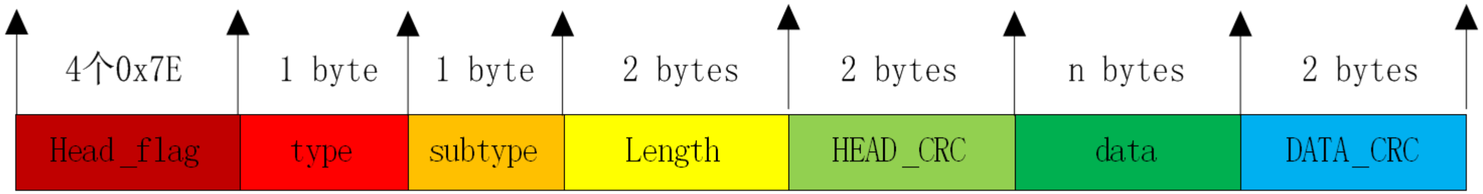
软件架构

Sensorhub 软件框架

- Overview
- HAL
- Kernel Driver
  - Data protocol
- CM4 Driver
- Dynamic Call

软件架构 -- Kernel Driver -- Data protocol

Sensorhub 架构中 AP Kernel Driver 与 CM4(CP)进行通信使用的通信协议数据单元为packet，其帧结构如下：



Head_Flag ：帧头，四个0x7E（魔数）；
Type ：sensor handle id（AP）；
Subtype ：sensor 的数据类型(CM4:sensor operation code id)；
Length ：data的长度；
HEAD_CRC ：帧头的CRC校验；
Data ：要发送的有效数据；
DATA_CRC ：data的CRC校验值；

注：协议相关的代码可参考shub_protocol.c (kernel4.4\drivers\iio\sprd_hub)

软件架构 -- Kernel Driver -- Data protocol

中间层

直接与硬件sensor ic通信的是运行于CM4内的ThreadX中的sensorhub驱动，AP OS linux kernel中基于iio subsystem的sensorhub驱动本质上是个中间层。该中间层的主要职能是：

1. shub_subtype_id
   将所有场景的控制和数据操作进行抽象，分配成枚举 id 的形式进行管理；

2. sys attributes
   将用户空间的命令和数据打包译码成帧数据 packet 并发送给 CM4；供用户空间读取一些简单的数据信息。

3. shub_read_thread
   读取 CM4 上传的帧数据 packet 并进行译码解析；

4. kfifo的数据存入和读取函数流程。

软件架构 -- Kernel Driver -- Data protocol

type

表示Android系统的sensor handle id。

定义： vendor\sprd\modules\sensors\libsensorhub\include\sensors\SensorHandleSync.h：

```
typedef enum {
    /*
     * Android Google type begin
     * WARNING: DO NOT CHANGE
     * */
    SENSOR_HANDLE_META_DATA                 = SENSOR_TYPE_META_DATA,                    /*  (0) */
    SENSOR_HANDLE_ACCELEROMETER             = SENSOR_TYPE_ACCELEROMETER,                /*  (1) */
    …
    SENSOR_HANDLE_PROXIMITY                 = SENSOR_TYPE_PROXIMITY,                    /*  (8) */
    ...
    SENSOR_HANDLE_WAKE_UP_BASE = 0x32,
    SENSOR_HANDLE_WAKE_UP_ACCELEROMETER             = SENSOR_HANDLE_WAKE_UP_BASE +
SENSOR_HANDLE_ACCELEROMETER,                /* (1)  */
    ...
    SENSOR_HANDLE_WAKE_UP_PROXIMITY                 = SENSOR_HANDLE_WAKE_UP_BASE +
SENSOR_HANDLE_PROXIMITY,                    /* (8)  */
    ...
}SENSOR_HANDLE_TYPE;
```

软件架构 -- Kernel Driver -- Data protocol

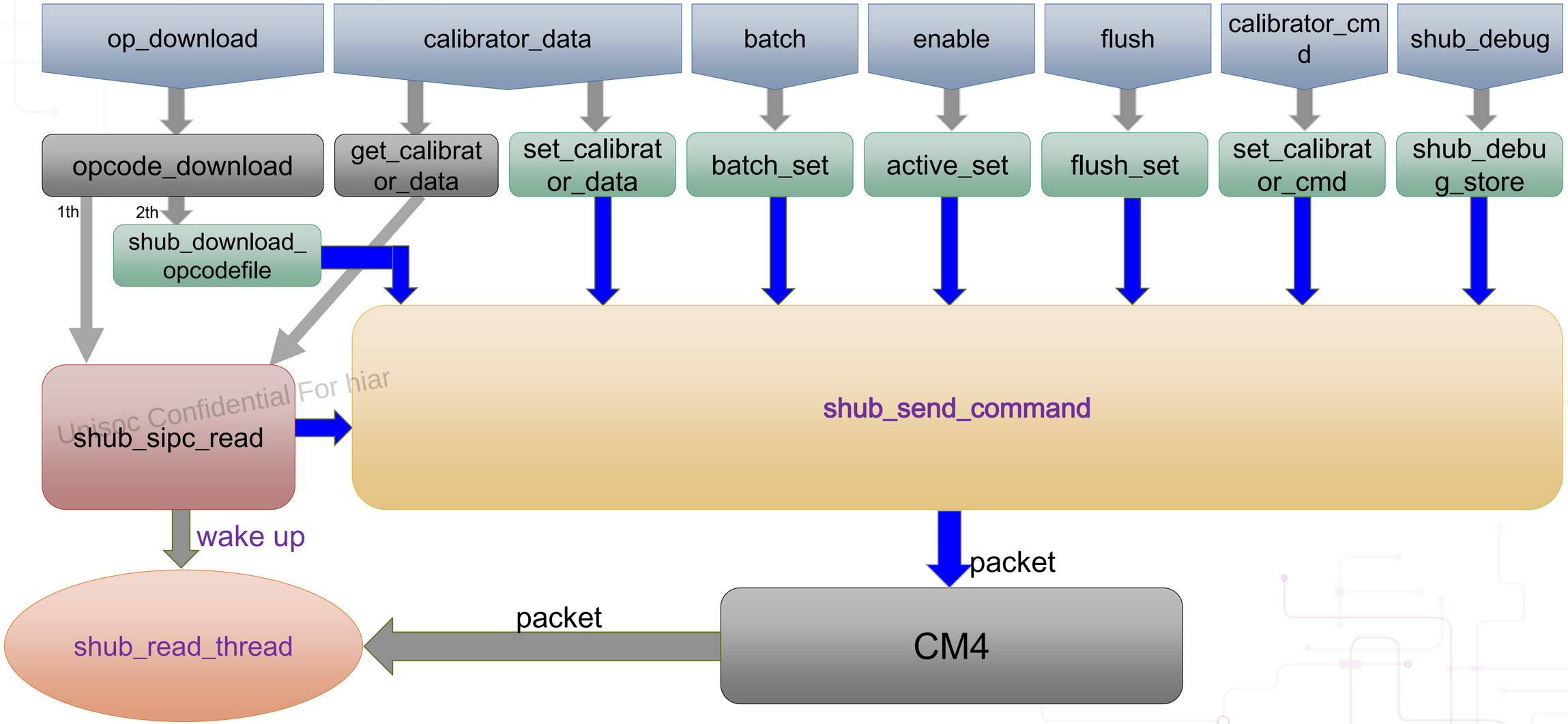
1. shub_subtype_id

```
enum shub_subtype_id {
    /* Android define */
    SHUB_SET_ENABLE_SUBTYPE,
    SHUB_SET_DISABLE_SUBTYPE,
    SHUB_SET_BATCH_SUBTYPE,
    SHUB_SET_FLUSH_SUBTYPE,
    /* pls add other android define here */
    /* Sprd define */
    SHUB_DATA_SUBTYPE,
    SHUB_LOG_SUBTYPE,
    SHUB_DOWNLOAD_OPCODE_SUBTYPE,
    SHUB_DOWNLOAD_CALIBRATION_SUBTYPE,
    SHUB_RESPONSE_SUBTYPE,
    /* read command begin */
    SHUB_GET_SENSORINFO_SUBTYPE,
    SHUB_SEND_MAG_CALIBRATION_FLAG,
    SHUB_SEND_DEBUG_DATA,
    SHUB_GET_MAG_OFFSET,
    /* other sprd define */
    /* 3rdAlgo define */
    SHUB_SET_CALIBRATION_DATA_SUBTYPE       =
THIRDALGO_START_SUBTYPE,
    SHUB_SET_CALIBRATION_CMD_SUBTYPE,
    SHUB_SET_TIMESYNC_SUBTYPE,
    SHUB_SET_HOST_STATUS_SUBTYPE,
    SHUB_GET_CALIBRATION_DATA_SUBTYPE,
    SHUB_GET_ENABLE_LIST_SUBTYPE,
    SHUB_GET_ACCELERATION_RAWDATA_SUBTYPE,
    SHUB_GET_MAGNETIC_RAWDATA_SUBTYPE,
    SHUB_GET_GYROSCOPE_RAWDATA_SUBTYPE,
    SHUB_GET_LIGHT_RAWDATA_SUBTYPE,
    SHUB_GET_PROXIMITY_RAWDATA_SUBTYPE,
    SHUB_GET_FWVERSION_SUBTYPE,
    SHUB_CWM_END_SUBTYPE                    =
THIRDALGO_END_SUBTYPE,  SHUB_END_SUBTYPE,
};
```

注：文件路径shub_protocol.h (kernel4.4\drivers\iio\sprd_hub)

软件架构 -- Kernel Driver

2. sys attributes



软件架构 -- Kernel Driver

2. sys attributes -- 将用户空间的命令和数据转换成帧数据packet发送给CM4

丰富的sys节点（包括iio subsystem提供的iio操作相关的），在唤醒读线程shub_read_thread的同时，将用户空间传来的命令和附加数据（如delay时间），按照帧协议转换成packet后发送给CM4。用户空间以此与内核驱动进行控制交互（如get_sensor_id）。

sys节点，由用户空间读写（例如libsensorhub）；
节点路径："/sys/class/sprd_sensorhub/sensor_hub"; 推荐直接操作符号链接"/d/sensor"，方便快捷。

sys节点对应的读（show）/ 写（store）函数

shub_sipc_read，用于单次、主动发起从CM4读取信息的场景。
支持的操作由SHUB_GET_XXX_SUBTYPE的shub_subtype_id来指定（SHUB_GET_MAG_OFFSET除外）。

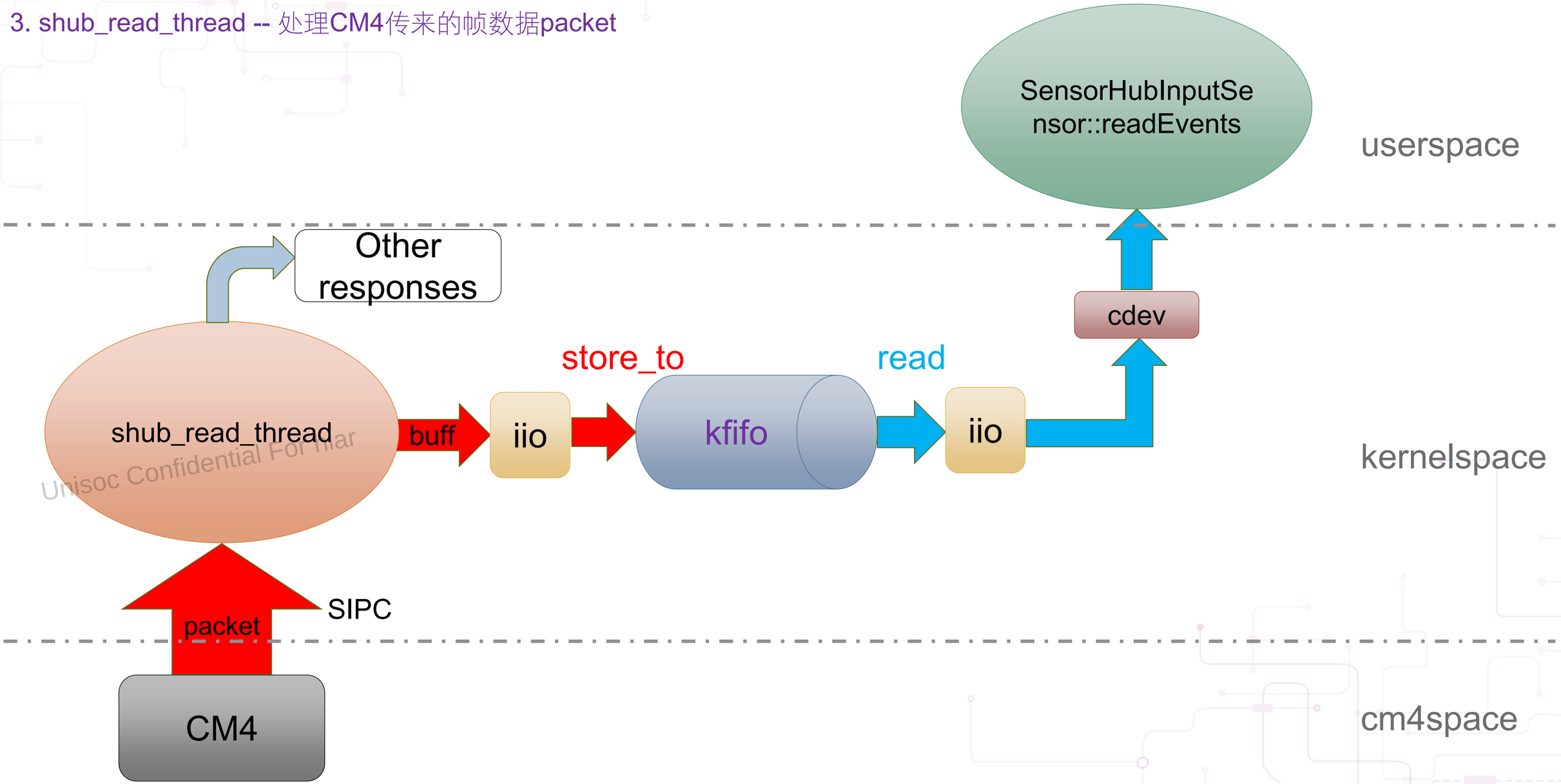
主要操作逻辑如下：

1. 将读条件reader_flag置1，唤醒正处于休眠等待的读线程shub_read_thread；
2. 紧接着，调用shub_send_command发送SHUB_GET_XXX_SUBTYPE命令给CM4，并进入休眠等待；
3. CM4在请求处理完毕后将rx_status置true，shub_sipc_read被唤醒执行结束。

shub_send_command，打包操作和数据发给CM4.
将操作和数据按照帧协议转换生成packet，写入PIPE通过SIPC的方式发给CM4处理.

软件架构 -- Kernel Driver

3. shub_read_thread -- 处理CM4传来的帧数据packet



软件架构 -- Kernel Driver

3. shub_read_thread -- 处理CM4传来的帧数据packet

线程shub_read_thread是sensorhub在linux kernel driver中的处理数据的核心。将CM4传来的packet按照协议解析出需要的data，然后对其进行分类处理，响应AP对CM4的请求。

3.1. SHUB_DATA_SUBTYPE – sensor raw data (buff) 上报处理

该操作请求，返回的data包含sensor id和sensor raw data，将其通过iio subsystem存入内核kfifo，与此同时用户空间线程从kfifo中读取（SensorHubInputSensor::readEvents()→InputEventCircularReader::fill()）。

注：

1) 系统启动，内核驱动加载时创建该线程并进入休眠等待；系统启动到Android层，HAL层SensorHubInputSensor类对象被创建时，构造函数中调用SensorHubInputSensor::downloadFirmware() 唤醒该线程。

2) sensor raw data 这种大批量数据的上报通过 cdev+kfifo的方式进行；

软件架构 -- Kernel Driver

3. shub_read_thread -- 处理CM4传来的帧数据packet

3.2. Other cases

- SHUB_SEND_DEBUG_DATA,
  cat shub_debug 调试节点读出的数据；

- SHUB_GET_MAG_OFFSET,
  保存CM4传来的sensor校准结果数据；

- SHUB_GET_XXX_SUBTYPE,
  唤醒发送操作请求完毕后陷入睡眠的等待队列，通知其接收需要的结果数据；

- SHUB_RESPONSE_SUBTYPE,
  表征命令已被CM4接受

- SHUB_SET_TIMESYNC_SUBTYPE
  AP 与 CM4 同步时间戳。

软件架构

Sensorhub 软件框架

- Overview
- HAL
- Kernel Driver
  - Data protocol
- **CM4 Driver**
- Dynamic Call

软件架构 – CM4 Driver

CM4 Driver运行于CM4内部的ThreadX中，主要功能如下：

1. 接收AP请求并解析处理

开启线程，轮询读取AP传来的命令请求数据帧packet，并进行解析处理；

2. 上报sensor数据给AP

响应操作请求，被enable之后，接收底层物理sensor上报的数据，通过算法处理后上报给AP。在此期间一些复合传感器的数据已经经过了算法融合处理。

3. 校准

目前支持的校准sensor有 acc, gyro, mag, proximity

4. 调用动态加载功能代码

软件架构

Sensorhub 软件框架

- Overview
- HAL
- Kernel Driver
  - Data protocol
- CM4 Driver
- **Dynamic Call**

软件架构 – Dynamic Call

Dynamic Call（动态加载）

对于当前sensorhub opcode 框架难以统一化处理的 sensor 问题，CM4提供了dynamic call（动态加载）机制来解决。这是针对CM4的补充插件功能。例如ESD检测恢复。

平台提供给客户patch_table_sensor.c文件，由客户与sensor FAE合作实现里面项目涉及sensor的enable， disable，get_data，updata_cfg函数(一般只需要实现get_data)。之后将patch_table_sensor.c文件返回展锐编译，展锐将bin发给客户验证。

关于 opcode 方案与动态加载方案的说明：

动态加载方案提供了 enable/update cfg/get data/disable，接口灵活。Opcode 方案与动态加载方案二者是串行执行顺序，先执行 opcode方案，再执行动态加载方案（如果存在）。以enable为例，先执行 opcode中eCmdEnableArray，再执行动态加载xxx_enable（如果存在）。

说明：动态加载机制详细操作说明详见后续 “客制化” 部分。

目录

客制化

一. 硬件

1. Design note: 关于 Sensorhub i2c 硬件设计注意事项

二. 软件

1. Opcode 客制化
2. 动态加载方案支持流程

客制化 -- 硬件

一. 硬件

1. Design note: 关于 Sensorhub i2c 硬件设计注意事项 -- 重要！

Sensorhub 架构设计中，sensor 作为i2c从设备，其挂靠的 i2c hardware interface 需要配置CP侧 i2c controller。而大部分其他 i2c 外设需配置 AP 侧 i2c controller 的，例如PMIC，Camera，NFC，TouchScreen等等。客户在硬件设计时，对于从属于 AP 侧的i2c设备，请勿与 Sensorhub sensor共用 i2c hardware interface，否则会导致冲突功能不可用，导致硬件改板。

客制化 -- 软件

二.软件

1. Opcode 客制

请参考如下文档：

《SC9863A sensor hub 客制化配置V1.4.pdf》（重要）

《SensorHub_配置文档V1.docx》（重要）

2. 动态加载方案支持流程

下面介绍展锐面向客户的的动态加载方案支持流程

2.1 AP opcode修改

```
static sensor_info_t sensorInfoConfigArray[] = {
    …
    .Vendor = 0xFE, //请把这里改成0xFE，使用动态加载
};
```

编译生成 out/…/vendor/lib64/libsensorsdrvcfg.so

客制化 -- 软件

2.2 动态加载文件修改

1）展锐提供源代码文件 patch_table_sensor.c 到客户；

2）参考《 dynamic_loading_user_guides.docx 》，实现需要的接口，并将修改完成的文件返回给展锐编译，展锐将编译生成的sensor_call_img.bin 给到客户验证。

说明：编译动态加载源代码文件需要 DS-5 编译器，此编译器需要付费购买，文档中有试用版的下载和安装使用说明。如果客户没有条件，可以将源代码文件提请展锐编译并释放相应bin文件。

2.3 快速验证

将 sensor_call_img.bin 重命名为 EXEC_CALIBRATE_MAG_IMAGE，再执行如下命令：

```
adb root && adb remount
adb push EXEC_CALIBRATE_MAG_IMAGE   vendor/firmware
adb push libsensorsdrvcfg.so vendor/lib64
adb reboot
```

客制化 -- 软件

2.4 验证成功请按如下方法移植：

1). sensor_call_img.bin拷贝到如下目录

vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/calibration/mag_cali/

2). 修改device/sprd/{platform}/{project}/{project}_Base.mk

```
#mag sensor cali
PRODUCT_COPY_FILES += \
- vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/calibration/mag_cali/
    akm_cali_img.bin:$(TARGET_COPY_OUT_VENDOR)/firmware/EXEC_CALIBRATE_MAG_IMAGE
+ vendor/sprd/modules/sensors/libsensorhub/ConfigSensor/calibration/mag_cali/
    sensor_call_img.bin:$(TARGET_COPY_OUT_VENDOR)/firmware/EXEC_CALIBRATE_MAG_IMAGE
```


3）请注意，多颗sensor的项目中如果多颗sensor都用到动态加载，实现全部集合在一支patch_table_sensor.c中。调试完毕后，客户请务必妥善维护 patch_table_sensor.c 文件，以便后续升级以及加入新的sensor调试内容。

说明：动态加载bin文件大小不能超过20KB，否则会编译失败。客户在进行代码实现时请尽量保持精简，如遇特殊情况请咨询展锐。

目录

校准

1. 校准相关请参考文档：

《sensor_hub_calibration_v1.0.docx》

《距感动态校准方案介绍V1.1.pdf 》

2. 目前软件在 “展锐工厂测试模式” 和 “开机工模” 都实现了 acc, gyro, mag, proximity sensor 的校准方案。

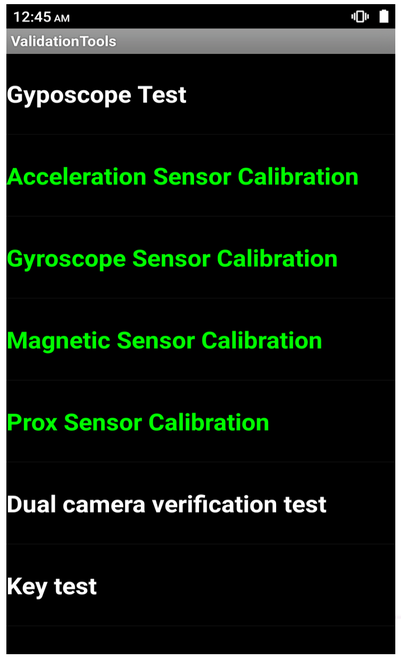
说明：

1) 展锐工厂测试模式（关机工模，Native MMI）：关机状态，按 Power Key + Volume-Up Key 进入；建议抽测项栏目中。

2) 开机工模（MMI test）：正常模式开机状态，输入*#*#83789#*#*进入。Item Test栏目中。

关机工模

开机工模



目录

Debug

一. 调试节点

1. Firmware

2. Runtime debug

二. Log

2.1 Log 抓取

2.2 Log 解析

三. 其他

3.1 adb 快速验证

Debug -- 调试节点 -- Firmware

一.调试节点

1. Firmware

File Node	File Type	Discription	Note
/sys/module/firmware_class/parameters/path	-	存储sensor firmware路径信息。	即“/data/vendor/sensorhub/”
/data/vendor/sensorhub/shub_fw_xxx	-	包含 sensorInfoConfigArray 和 sSensorOpcode。	可以通过文件名检查项目 BoardConfig.mk 中实际添加的 sensor 名称
/data/vendor/sensorhub/shub_fw_xxx_cali	-	包含sensor 静态校准参数 eXxxCaliDataArray。	用途同上。 注意：当前sensorhub架构中只需配置 proximity sensor的静态校准参数，其他 sensor不用关心。
/sys/module/shub_core/parameters/xxx_firms	-	支持的sensor类型和名称	查看sensor名称，以及是否有多颗兼容料。
/sys/module/shub_core/parameters/sensor_fusion_mode	-	Acc，gyro，mag的融合类型	

Debug -- 调试节点 -- Runtime debug

一.调试节点

2. Runtime debug

File Node	File Type	Discription	Note
/d/sensor/shub_debug	-	实时读写 sensor 寄存器（重要）	详细操作方法见另一页
/d/sensor/hwsensor_id	-	所有Sensor 类型，状态，id和名字。 格式如下： Sensor type； Status: 0 not ready; 1 ok; 2 fail; Id：Vendor id(high 8 bits) + Chip id(low 8 bits)； Sensor name	Ex: prox:1(0x0092, ltr553)
/d/sensor/sensor_info	-	同 hwsensor_id，简化版	
/d/sensor/logctl		控制kernel层驱动log输出	详细操作方法见另一页
/dev/sctl_pm	-	CM4 sensorhub log 开关（重要）。 可以直接看到贴近sensor硬件的操作信息。	adb shell $ su # echo log on > /dev/sctl_pm # cat /dev/slog_pm

Debug -- 调试节点-- Runtime debug

一.调试节点

2. Runtime debug

节点：/d/sensor/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

节点：/d/sensor/logctl

Description:

echo "bit val " > logctl

Detail:

bit: [0 ~ 7]

bit0: debug message

bit1: sysfs operate message

bit2: function entry message

bit3: data path message

bit4: dump sensor data

val: [0 ~ 1], 0 means close, 1 means open

bit [8] control all flag, all open or close

说明：节点操作说明亦可通过 cat 该节点获取

Debug -- Log -- Log抓取

二. Log

2.1 Log抓取

实际解决问题当中，ylog是最常用。除了通用的ylog抓取方式外，有三点注意事项：

1）Log level

2）cm4 sensorhub log的存储策略；

3）sensor 初始化 log。

2.1.1 Log level

HAL层可以客制化 log的输出，方法如下：

SensorHubDynamicLogel.cpp (vendor\sprd\modules\sensors\libsensorhub)

```
/* verbose log messages */
uint32_t DBG_VERBOSE = 0;
/* log entry in all one-time functions */
uint32_t FUNC_ENTRY = 0;
/* log the data input from the events */
uint32_t INPUT_DATA = 0;
/* log sysfs interactions as cat/echo for repro purpose on a shell */
uint32_t SYSFS_VERBOSE = 0;
/* log the data fetched from the handlers */
uint32_t DUMP_DATA = 0;
/* process log messages */
uint32_t PROCESS_VERBOSE = 0;
/* log entry in all handler functions */
uint32_t HANDLER_ENTRY = 0;
/* log some a lot more info about the internals */
uint32_t ENG_VERBOSE = 0;
```

Debug -- Log -- Log抓取

二. Log

2.1.2 cm4 sensorhub log的存储策略

- 未插SD卡

ylog：

/storage/emulated/0/ylog

AP侧sensorhub log，包含 android和kernel部分。

cm4 sensorhub log：

/data/vendor/ylog/sensorhub

CM4侧sensorhub log，包含CM4 ThreadX sensor chip driver部分。

- 插SD卡

此时cm4 sensorhub log也被整合到ylog中： /storage/sdcard0/ylog。

注意：

1) 使用SD卡无法抓取到开机初始化过程的 cm4 sensorhub log，关于此部分log请参考下面2.1.3部分。

2) USER 版本如果抓完 log 要导出，可插 SD 卡后，进入系统设置打开开发者选项并使用命令“adb shell log_ctr slogmodem COLLECT_LOG” 导到 SD卡 ylog目录。

Debug – Log -- Log抓取

二. Log

2.1.3 sensor 初始化 log

通常情况下客户提供的cm4 sensorhub log只包含系统起来之后sensor log，当遇到一些复杂问题时，这是不够的，此时需要不插SD卡使用如下方法抓取cm4 侧 sensor 初始化的 log（含 opcode download和check id）：

1) 清空再打开ylog
2) 选择 Power-Off 的方式关机再手动开机抓取ylog，直接 Restart 重启抓取的ylog中无法抓取这部分信息。

Debug – Log -- Log解析

2.2 Log 解析

1）查看HAL层距感接近离开, 校准底噪值和实时psensor光强值

adb shell

logcat | grep "processTypeAEvent::Proximity" –i

实例：

logcat | grep " processTypeAEvent::Proximity " -i

01-02 18:50:41.093 3611 3689 D SensorHub: SPRD[761]processTypeAEvent::Proximity sensor x:5.000000 y:0.000000 z:36.000000

01-02 18:50:42.446 3611 3689 D SensorHub: SPRD[761]processTypeAEvent::Proximity sensor x:0.000000 y:0.000000 z:231.000000

01-02 18:50:43.136 3611 3689 D SensorHub: SPRD[761]processTypeAEvent::Proximity sensor x:5.000000 y:0.000000 z:40.000000

01-02 18:50:43.836 3611 3689 D SensorHub: SPRD[761]processTypeAEvent::Proximity sensor x:0.000000 y:0.000000 z:110.000000

解析：

x: 5.000000 远离；0.000000 接近；

y: 0.000000 为工厂校准得到的底噪值，为0说明没有经过校准，假如校准得到的底噪值大于190，说明机器结构有问题

z: 36.000000 psensor光强值

Debug – Log -- Log解析

2.2 Log 解析

2）dump sensor data log

adb shell
echo 4 2 > /sys/class/sprd_sensorhub/sensor_hub/logctl
logcat | grep "dynamicLogerDumpData" -i
注意：一定要加-i参数，否则无log吐出

实例：
D SensorHub: SPRD[83]dynamicLogerDumpData::Light Sensor:
D SensorHub: data: 109.000000, 0.000000, 109.000000 timestamp:181650840000000
D SensorHub:
. . .
D SensorHub: SPRD[83]dynamicLogerDumpData::Accelerometer Sensor:
D SensorHub: data: -7.873866, -0.739746, 6.959358 timestamp:181650749000000
D SensorHub:
上面这条是将所有支持的sensor数据打印出来，实际操作中我们更希望查看某一指定sensor的数据：

实例：
logcat | grep "dynamicLogerDumpData::Accelerometer" -i
D SensorHub: SPRD[83]dynamicLogerDumpData::Accelerometer Sensor:
D SensorHub: data: -7.873866, -0.739746, 6.959358 timestamp:181650749000000
D SensorHub:

解析：
data: -7.873866, -0.739746, 6.959358 acc sensor的x y z数据
timestamp:181650749000000 该笔数据的时间戳

Debug – Log -- Log解析

2.2 Log 解析

3) 查看kernel sensorhub log

adb shell

cat /proc/kmsg | grep "sensorhub" -i

实例：

cat /proc/kmsg | grep "sensorhub" -i

<6>[1038.016756] c0 3826 [SensorHub]active_set 902:buf=1 0

<6>[1038.016779] c0 3826 [SensorHub]active_set 909:handle = 1, enabled = 0

<6>[1040.962249] c1 4320 [SensorHub]active_set 902:buf=3 0

<6>[1040.962272] c1 4320 [SensorHub]active_set 909:handle = 3, enabled = 0

解析：

handle：sensor id

enabled：1 enable; 0 disable

说明：handle 参考 vendor\sprd\modules\sensors\libsensorhub\include\sensors\SensorHandleSync.h

Debug – Log -- Log解析

2.2 Log 解析

4) cm4 sensorhub log

实例：

0x64ccda9a:HandleCommandRequest ----- type:0x1, subType:0x2

...

0x64ccdaaa:HandleCommandRequest ----- type:0x1, subType:0x0

...

0x64ce4437:HandleCommandRequest ----- type:0x1, subType:0x1

解读：

type: sensor id；0x1表示acc sensor

subType: operation id；0x2 batch，0x0 enable，0x1 disable

说明：

type 同 handle id，详细参看 vendor\sprd\modules\sensors\libsensorhub\include\sensors\SensorHandleSync.h；

subtype 详细参看 kernel4.4\drivers\iio\sprd_hub\shub_protocol.h

Debug – Log -- Log解析

2.2 Log 解析

4) cm4 sensorhub log

实例：

[计算时间间隔]

0x4b3960:SENSORHUB:DRIVER_COMM_ACC_Enable

0x4b396c:SENSORHUB:DRIVER_COMM_ACC_SetRate the odr is 20

解析：

CM4 log trace 函数使用了 32KHZ 的时钟频率，即 10^9 ns /(32 * 1024) = 30517 ns = 31 us

这两行 log的时间间隔为： (0x4b396c - 0x4b3960) * 31us = 372 us

Debug – 其他

三. 其他

3.1. adb push 快速验证

实际调试过程当中，我们修改的文件可能很少，这个时候可以选择一些快速的调试方法 来提高效率。

out：obj/SHARED_LIBRARIES/libsensorsdrvcfg_intermediates/libsensorsdrvcfg.so

--- 对应vendor\sprd\modules\sensors\libsensorhub**ConfigSensor**中的修改（**opcode**）

out：obj/SHARED_LIBRARIES/libsensorlistcfg_intermediates/libsensorlistcfg.so

--- 对应vendor\sprd\modules\sensors\libsensorhub**ConfigFeature**中的修改

out：obj/SHARED_LIBRARIES/sensors.sp9863a_intermediates/sensors.sp9863a.so

-- 对应vendor\sprd\modules\sensors**libsensrohub**目录下平台通用文件

adb push libsensorsdrvcfg.so vendor/lib64

adb push libsensorlistcfg.so vendor/lib64

adb push sensors.sp9863a.so vendor/lib64/hw

声明

本文件所含数据和信息都属于紫光展锐所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能保证。本文件中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供内部参考，若需要商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。 当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。除合同另有书面约定外，紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与本文件相关的直接或间接的、任何伤害或损失。

THANKS