一. 压力采集接口

1.1. start\_pressure\_curve---->set\_press\_reg

sem\_init

pthread\_create

signal(SIGIO,read\_adc\_data)

open() 打开从设备

start\_fpga 启动开始采集

set\_press\_reg ()

配置采集板，包括上电，打开主设备，向主设备下发各种寄存器

sem\_init(&bin\_sem, 0, 0)

初始化一个信号量，用于采集到64K数据后，进入小线程中计算用

pthread\_create(&id, NULL, press\_thread\_curve, (void\*)&data);

本地自创建一个线程，用于处理计算采集到的数据

press\_thread\_curve(): 将采集的64K数据进行算法运算，计算后的数据通过JNI回调函数回调给JNI层

signal(SIGIO,read\_adc\_data)

接收驱动层的异步信号，调用read\_adc\_data开始采集数据

read\_press\_data(): 采集64K数据后，sem\_post()信号

1.2. stop\_sample--->sem\_post(&bin\_sem) 关闭创建的线程

ioctl 停止驱动层DMA采集

stop\_fpag

sem\_destory()内部信号量

close(fd) 关闭设备

spi\_poweroff() 下电

1.3. framework-->JNI--->HAL层接口连接：

startNativeSingleAD ---> android\_debug\_JNITest\_startSingleAD ---> start\_ pressure\_dial //开始表盘模式

startNativeGroupAD ---> android\_debug\_JNITest\_startGroupAD ---> start\_ pressure\_curve //开始曲线模式

startNativeCaliAD ---> android\_debug\_JNITest\_startCaliAD---> start\_ pressure\_flag0 //开始标0模式

stopNativeAD ---> android\_debug\_JNITest\_stopAD---> stop\_ sample //停止采集

startNativeInit---> android\_debug\_JNITest\_startNativeInit ---> spi\_get\_spi\_interface //初始化

getNativeADFrequency---> android\_debug\_JNITest\_getNativeADFrequency---> spi\_get\_freq //获取采集频率

二.振动采集参数接口定义

2.1.定义时域波形结构体

typedef struct time\_wave\_para

{

int data\_type;//数据类型

int signal\_type; //信号类型

float min\_freq; //下限频率

float max\_freq; //上限频率

int wave\_length; //波形长度

int range\_mode;//量程方式

int range\_accel\_value;//加速度量程

int range\_speed\_value; //速度量程

int range\_disp\_value; //位移量程

float range\_gain\_value1;//一级增益

float range\_gain\_value2;//二级增益

int trig\_mode; //触发方式

float trig\_value;//触发电平值

}timewave;

2.2.HAL层向JNI层提供的接口如下：

启动采集：

static int start\_vibrate\_CH\_timewave(struct spictl\_device\_t\* dev, int ch\_num,struct time\_wave\_para wave )

static int start\_vibrate\_CH\_totalrend(structspictl\_device\_t\* dev, int ch\_num,struct total\_rend\_para wave )

static int start\_vibrate\_evalute\_level( struct spictl\_device\_t\* dev , struct time\_wave\_para tWave );

参数说明：

dev: HAL层对外的设备结构体

ch\_num: 输入值，表示具体通道个数，1表示单通道采集，2表示双通道采集

wave: 输入值，表示数据类型的结构体(时域波形，总值趋势)

停止采集：

static int stop\_sample(struct spictl\_device\_t\* dev)

参数说明：

dev: HAL层对外的设备结构体

时域转频域及15个特征值：

static float\* spi\_get\_feature\_value( struct spictl\_device\_t\* dev , float pData[] , int data\_len );

static float\* spi\_time\_to\_freq\_value( struct spictl\_device\_t\* dev , float pData[] , int data\_len );

参数说明：

pData： 上层APP下传的数据数组

data\_len： 下发的有效数组长度

2.3.主设备写寄存器的函数如下：

enable\_CH\_A(true);// ture:表示使能CH1, false: 不使能CH1

enable\_CH\_A\_integrate(true);// ture:使能CHA的积分, false: 不使能CHA的积分

set\_adc\_mode(HighSpeed);//设置ADC的工作模式 ,分HighSpeed,HighPrecision,LowPower,LowSpeed四种模式

set\_adc\_clk\_rate(para.adc\_clk);//设置ADC时钟速率

set\_couple\_mode(CHA, DC);//设置对应通道耦合方式， 分AC or DC

set\_sample\_type(CHA, VOLTAGE);//设置对应通道采集类型 , 分电压or电流

set\_24V(CHA, false);//设置对应通道是否启用24v电源的开关激励 ，ture：表示启用激励，false:表示不启用激励

set\_voltage\_range(CHA, V25);//设置对应通道电压量程, 分V25 ,V2.5, V0.25 三种电压量程

set\_integrate(CHA, SHIFT);//设置对应通道积分选择, SHIFT:表示位移，SPEED:表示速度

set\_triger\_ch(CHA);//设置模拟触发通道 ，CHA:表示采用通道1，CHB:表示采用通道2

set\_triger\_mode(Manual);//设置开始采集的触发方式, 分Manual,RTriger,AnalogRising,AnalogFalling四种方式

set\_triger\_threshold(para.trig\_value);//设置触发阈值----

以时域波形函数为例：

static int start\_vibrate\_CH\_timewave(struct spictl\_device\_t\* dev, float flag0\_value, int ch\_num,struct time\_wave\_para wave )

{

g\_chNum = ch\_num;

ALOGD("start\_vibrate\_CH\_timewave=====ch\_num = %d",g\_chNum);

if(g\_chNum == SINGLE\_CH)

set\_singleCH\_reg();//设置单通道采集寄存器

if(g\_chNum == DOUBLE\_CH)

set\_doubleCH\_reg();//设置双通道采集寄存器

sig\_res = sem\_init(&bin\_sem, 0, 0);

if (sig\_res != 0)

{

ALOGD("sig\_res initialization failed");

}

vibration\_flag0\_value = flag0\_value;

my\_timewave = wave;

g\_waveLength = my\_timewave.wave\_length;

pthread\_create(&id, NULL, time\_wave\_thread, (void\*)&my\_timewave);

signal(SIGIO, read\_vibrate\_data);

common\_func();

return 0;

}

启动采集函数中会针对通道个数不同，调用不同的寄存器设置函数配置主寄存器，后创建对应波形结构体线程。

time\_wave\_thread线程会根据不同的通道数和其它参数，调用不同的算法函数

read\_vibrate\_data 根据传下来的结构体中波形长度，进行原始数据采集

下限频率： 0.16HZ , 1HZ, 2HZ, 5HZ, 10HZ, 20HZ, 50HZ, 100HZ

上限频率： 500HZ, 1KHZ, 2KHZ, 2.5KHZ, 4KHZ, 5KHZ, 10KHZ, 20KHZ, 40KHZ

三.数据回调接口定义：

3.1. JNI层数据回调接口

SpiVibrateCallbacks mSpiCb2 = { //总的回调接口

request\_single\_callback,

request\_stop\_callback,

};

static void request\_single\_callback(float data[],int length, bool isCollectData) //单通道正常采集回调数据

{

request\_vibration\_single\_callback(data, length, isCollectData);

}

参数说明：

data[]: 需要回调的有效数据

length: 回调的有效数据长度

isCollectData: 回调数据是否有效

static void request\_stop\_callback(bool isStop)//停止回调接口

{

request\_vibration\_stop\_callback(isStop);

}

参数说明：

isStop：表示回调接口的状态

3.2. HAL层调用JNI接口

typedef struct {

spictl\_single\_ch\_callback single\_ch\_callback;

spictl\_stop\_ch\_callback stop\_ch\_callback;

} SpiVibrateCallbacks;

static SpiVibrateCallbacks vibrate\_callback\_backup; //振动回调接口

实例用法：

vibrate\_callback\_backup.single\_ch\_callback( time\_CH1\_smp\_buf , g\_waveLength , true ); /////回调时域波形

vibrate\_callback\_backup.stop\_ch\_callback( false);