VC XX（LC 1X）系列芯片应用指南

V 0.1

2021.07.13

****说明****

本文档适用于成都维客昕微 VC XX（LC 1X）系列心率血氧传感器，具体型号为：VC31B，VC32S，VC52S，VC9201，LC11，LC10A，LC11S，其中 VC31B 以及 LC11、LC10 为单心率传感器，不带血氧功能；VC32S，VC52S，VC9201，LC11S 为心率血氧传感器，另 LC 1X 芯片的 C 程序接口函数名与其他芯片的接口函数名有区别，但调用流程没有任何区别。

版本记录

版本号	修订人	修订日期	修订内容
V 0.1	ZSL	20210713	初版
V 0.2	ZSL	20210908	增加适用本文档的其他芯片型号，并增加目录

目录

1. 主控资源要求

1.1 通信（IIC）

VC XX（LC 1X）系列芯片仅支持 IIC 通信，不支持 SPI 通信，IIC 不支持设备地址配置，IIC 7bit 地址为 0x33，8bit 地址为 0x66/0x67。

1.2 外部中断

VC XX（LC 1X）系列芯片会因不同中断原因从中断脚给出中断信号，主控需要响应外部事件中断，调用 vcHr02_process 函数（LC 1X 系列芯片为 vcHr11_process 函数）进行处理，不支持定时查询。

1.3 RTC/通用定时器

VC XX（LC 1X）系列芯片需要校准内部时钟，需要返回 RTC 或者通用定时器的计数寄存器的计数值，计数频率在 30KHz 附近最优，最低不能低于 10KHz，且要求计数器为向上计数。

（特别说明：返回值不是定时的时间，而是计数值，例如某一定时器的计数寄存器为 32 位，计数频率为 32K，则计数寄存器里的计数值每 1/32000s 会加 1，从 0 计数到「2^{32}-1」后，清 0，重新计数）

以上为 VC XX（LC 1X）系列芯片需要的主控上的全部资源，如果主控有外部时钟源（RTC），最好提供 RTC 的计数值。

2. 函数接口说明

操作 VC XX（LC 1X）系列芯片需要使用配套的 C 程序库，以下仅提供客户需要使用到的 C 程序函数接口说明，其余代码客户请勿自行修改。

2.1 VC XX（LC 1X）控制接口函数

2.1.1 vcHr02StartSample

函数原型	vcHr02Ret_t vcHr02StartSample(vcHr02_t *pvcHr02)
参数	pvcHr02：包含工作模式，佩戴状态等变量的结构体

返回值	
功能说明	开启测量，寄存器初始化完后开启测量或调用 vcHr02StopSample 函数后开启
备注	

2.1.2 vcHr02StopSample

函数原型	vcHr02Ret_t vcHr02StopSample(vcHr02_t *pvcHr02)
参数	pvcHr02：包含工作模式，佩戴状态等变量的结构体
返回值	
功能说明	停止测量，芯片进入休眠，调用 vcHr02StartSample 退出休眠，开启测量
备注	参数无用

2.1.3 vcHr02SoftReset

函数原型	vcHr02Ret_t vcHr02SoftReset(vcHr02_t *pvcHr02)
参数	pvcHr02：包含工作模式，佩戴状态等变量的结构体
返回值	
功能说明	芯片软复位，可不用断电复位芯片，调用后，若芯片重新工作需要重新初始化
备注	

2.2 用户实现接口函数

2.2.1 vcHr02WriteRegisters

函数原型	vcHr02Ret_t vcHr02WriteRegisters(uint8_t startAddress, uint8_t *pRegisters, uint8_t len)
参数	startAddress：IIC 写入的寄存器起始地址 pRegisters：用于缓存 IIC 写入的字节内容的指针 len：写入的数据的长度（字节）
返回值	
功能说明	IIC 写函数，向寄存器写入数据，由用户实现
备注	需要支持多字节写

2.2.2 vcHr02ReadRegisters

函数原型	vcHr02Ret_t vcHr02ReadRegisters(uint8_t startAddress, uint8_t *pRegisters, uint8_t len)
参数	startAddress：IIC 读取的寄存器起始地址 pRegisters：用于缓存从 IIC 读取的字节流的指针 len：需要读取的数据的长度（字节）
返回值	
功能说明	IIC 读函数，从寄存器读取数据，由用户实现
备注	需要支持多字节读

2.2.3 vcHr02GetRtcCountFromMCU

函数原型	uint32_t vcHr02GetRtcCountFromMCU(void)
参数	无
返回值	RTC 或通用定时器的计数寄存器的当前计数值
功能说明	从 MCU 获取 RTC 或通用定时器的计数寄存器的当前计数值，用于校准内部时钟，由用户实现
备注	计数频率在 30KHz 左右最优，最低不低于 10KHz

2.3 初始化及中断处理函数

2.3.1 vcHr02Init

函数原型	void vcHr02Init(vcHr02_t *pVcHr02,vcHr02Mode_t vcHr02WorkMode)
参数	pVcHr02：芯片配置项结构体，包括工作模式，采样率等参数 vcHr02WorkMode： VCWORK_MODE_HRWORK：心率测量模式 VCWORK_MODE_CROSSTALKTEST：漏光测试模式 VCWORK_MODE_SPO2WORK：血氧测量模式
返回值	

功能说明	初始化芯片
备注	以上为三种常规工作模式，其中血氧模式部分芯片不支持（没有血氧测试功能），漏光测试模式为厂测模式，必须添加

2.3.2 vcHr02_process

函数原型	void vcHr02_process(AlgoSportMode_t vcSportMode)

参数	vcSportMode：运动类型参数，支持的运动类型在心率算法头文件 algo.h 中
	SPORT_TYPE_NORMAL = 0x00 // 日常
	SPORT_TYPE_RUNNING = 0x01 // 跑步
	SPORT_TYPE_RIDE_BIKE = 0x02 // 骑行
	SPORT_TYPE_JUMP_ROPE = 0x03 // 跳绳
	SPORT_TYPE_SWIMMING = 0x04 // 游泳
	SPORT_TYPE_BADMINTON = 0x05 // 羽毛球
	SPORT_TYPE_TABLE_TENNIS = 0x06 // 乒乓球
	SPORT_TYPE_TENNIS = 0x07 // 网球
	SPORT_TYPE_CLIMBING = 0x08 // 爬山
	SPORT_TYPE_WALKING = 0x09 // 徒步
	SPORT_TYPE_BASKETBALL = 0x0A // 篮球
	SPORT_TYPE_FOOTBALL = 0x0B // 足球
	SPORT_TYPE_BASEBALL = 0x0C // 棒球
	SPORT_TYPE_VOLLEYBALL = 0x0D // 排球
	SPORT_TYPE_CRICKET = 0x0E // 板球
	SPORT_TYPE_RUGBY = 0x0F // 橄榄球
	SPORT_TYPE_HOCKEY = 0x10 // 曲棍球
	SPORT_TYPE_DANCE = 0x11 // 跳舞
	SPORT_TYPE_SPINNING = 0x12 // 动感单车
	SPORT_TYPE_YOGA = 0x13 // 瑜伽
	SPORT_TYPE_SIT_UP = 0x14 // 仰卧起坐
	SPORT_TYPE_TREADMILL = 0x15 // 跑步机
	SPORT_TYPE_GYMNASTICS = 0x16 // 体操
	SPORT_TYPE_BOATING = 0x17 // 划船
	SPORT_TYPE_JUMPING_JACK = 0x18 // 开合跳
	SPORT_TYPE_FREE_TRAINING = 0x19 // 自由训练

返回值	
功能说明	中断服务函数，处理中断事件
备注	以上运动类型有部分暂未适配，未适配类型传 0x00，LC 1X 系列芯片的算法无运动类型参数

3. 调试流程

3.1 芯片选择

VC XX 系列芯片驱动共用一套代码，但寄存器配置不同，所以调试流程的第一步需要根据客户所使用的具体芯片型号，在驱动代码中选择对应的芯片型号。LC 1X 系列芯片驱动与 VC XX 系列芯片驱动仅函数和部分变量命名不同，执行逻辑基本保持一致（只有部分逻辑做调整），调试流程完全一样，LC 1X 系列芯片中 LC10A 不需要选择芯片类型，会自动读取 ID 区分，使用 LC10A 芯片，忽略本步骤即可。

芯片类型选择在文件 vcHr02Hci.h 中，对应芯片的宏定义置 1，其余置 0，如下图所示为选择 VC52S：

module_heart_vc_int.c | vcHr02Hci.c | vcHr02Hci.h

```
22      If your chip is VC31B/VC51B, please comment out these two codes.
23      如果用的芯片是VC31B/VC51B，请注释掉下面两句包含头文件的代码。
24  */
25  #include "spo2Algo.h"
26  #include "vcSportMotionAlgo.h"
27
28
29  /**************************************************************************
30  *
31  *                          Register Address Declaration
32  *
33  **************************************************************************/
34  #if   1
35                                          // Chose your chip model //
36  /* Read Only */                         //     选择芯片型号      //
37  #define  VCHR02ISVC31B          0
38  #define  VCHR02ISVC32S          0
39  #define  VCHR02ISVC51B          0
40  #define  VCHR02ISVC52S          1
41  #define  VCHR02ISVC52T          0
42  #define  VCHR02ISVC9201         0
43
```

图 - 1

3.2 IIC 初始化及读写函数实现

客户需实现 IIC 的初始化代码，并在 module_heart_vc_int.c 文件中 vcHr02WriteRegisters 和 vcHr02ReadRegisters 函数下实现 IIC 读写代码，要求能连续读写任意字节数，因底层驱动 IIC 读写操作均用该读写函数，所以不能改变函数名称。IIC 7bit 地址为 0x33，8bit 地址为 0x66/0x67。实现 IIC 读写函数后，可读 0x00 寄存器地址，获得 Chip ID 为 0x21（LC10A 为 0x29）。

IIC 读写函数实现位置如下图所示：

```
/*
 * @brief  Register writing.
 *
 *
 **/
vcHr02Ret_t vcHr02WriteRegisters(uint8_t startAddress, uint8_t *pRegisters, uint8_t len)
{
/*********************************************************************************
* @param
*       startAddress:Register address.
*       pRegisters  :Point to the data to write.
*       len         :Byte to be transmitted.
*
*
*       7bit I2C own address:0x33
*       8bit I2C own address:0x66(the least significant bit is the W/R control bit)
*********************************************************************************/

/*------------------INSERT YOUR CODE HERE-----------------*/

    return VCHR02RET_ISOK;
}
```

图 - 2

```
/*
 * @brief  Register reading.
 *
 *
 **/
vcHr02Ret_t vcHr02ReadRegisters(uint8_t startAddress, uint8_t *pRegisters, uint8_t len)
{
/*********************************************************************************
* @param
*       startAddress:Register address.
*       pRegisters  :Point to the location to save.
*       len         :Byte to be transmitted.
*
*
*       7bit I2C own address:0x33
*       8bit I2C own address:0x67(the least significant bit is the W/R control bit)
*********************************************************************************/

/*------------------INSERT YOUR CODE HERE-----------------*/

    return VCHR02RET_ISOK;
}
```

图 - 3

3.3 RTC/定时器计数值获取函数实现

在 module_heart_vc_int.c 文件中 vcHr02GetRtcCountFromMCU 函数下实现获取 RTC 或定时器计数值的代码，直接返回计数值。计数频率在 30KHz 附近最优，最低不能低于 10KHz，且要求计数器为向上计数。

返回值不是定时的时间，而是计数值，例如某一定时器的计数寄存器为 32 位，计数频率为 32K，则计数寄存器里的计数值每 1/32000s 会加 1，从 0 计数到「2^{32}-1」后，清 0，重新计数。计数值获取函数实现位置如下图所示：

```
module_heart_vc_int.c | vcHr02Hci.c | vcHr02Hci.h
447
448 /*
449  * @brief  Get the count value of RTC or general timer of MCU.
450  *
451  *
452  **/
453 uint32_t vcHr02GetRtcCountFromMCU(void)
454 {
455 /*------------------INSERT YOUR CODE HERE-----------------*/
456
457 /*
458     return RTC_GetCounter();
459 */
460 }
461
```

图 - 4

根据客户使用的 RTC 或定时器的实际计数频率修改 module_heart_vc_int.c 文件中变量 mcuOscData 的值，例如，计数频率为 30K，则变量 mcuOscData = 30000。

3.4 调用流程

在选择对应芯片，并实现了 IIC 读写函数和 RTC/定时器计数值获取函数之后，将 module_heart_vc_int.c 中的 main 函数删除。

1. 调用 IIC，中断初始化；
2. 调用 vcHr02Init(&vcHr02,vcMode)初始化模块，变量 vcHr02 及变量 vcMode 已定义，vcMode 为模块的工作模式（即心率，血氧，漏光测试模式等，默认为心率模式），客户可通过改变该变量的值切换模块至不同工作模式，调用该函数之后，模块会亮灯；
3. 在模块中断服务函数中将 vcHr02IRQFlag 置为 true，并将中断处理函数 vcHr02_process(vcSportMode)放入任务队列。

以上三步之后，若模块对空时灯会熄灭（摘下），手指贴近模块灯会亮起（佩戴），且手指靠近时，灯会变暗，手指远离时，灯会变亮，则证明模块已正常工作，驱动调试完成。接下来可以验证出值情况，心率模式下算法输出的心率值以及血氧模式下算法输出的血氧值分别在 vcHr02_process 函数中如下图-5、图-6 所示位置，心率出值时间在 10 秒至 20 秒，不同个体因信号差异会有所不同。血氧模式包含血氧算法以及血氧运动量判断算法，所以真实血氧值的取得需要判断算法输出的血氧值是否大于 0 且运动量标志是否为 0（运动量标志输出 0，血氧测量过程中运动幅度小，输出血氧值可信度高）。

```
                algoInputData.envSample = vcHr02.sampleData.envValue[0];
                for(algoCallNum= 0; algoCallNum < ppgLength; algoCallNum++)
                {
                    algoInputData.ppgSample = vcHr02.sampleData.ppgValue[algoCallNum];
                    algoInputData.axes.x = xData[algoCallNum];//The direction vertical with ARM.
                    algoInputData.axes.y = yData[algoCallNum];//The direction parallel with ARM.
                    algoInputData.axes.z = zData[algoCallNum];//The direction upside.
                    Algo_Input(&algoInputData, 1000/vcHr02SampleRate, vcSportMode); 算法输入

                }

                Algo_Output(&algoOutputData); 算法输出
                HeartRateValue = algoOutputData.hrData;  心率值

                if(HeartRateValue == -1)
                {
                    Algo_Init();
                }
            }
        }
```

图 - 5

```
            for(algoCallNum= 0; algoCallNum < ppgLength/2; algoCallNum++)
            {
                vcIrPPG = vcHr02.sampleData.ppgValue[algoCallNum*2];
                vcRedPPG = vcHr02.sampleData.ppgValue[algoCallNum*2+1];
                vcSpo2Value = vcSpo2Calculate(vcRedPPG,vcIrPPG);血氧算法输入即输出
#if GsensorEn
                vcSportFlag = vcSportMotionCalculate(xData[algoCallNum], yData[algoCallNum],zData[algoCallNum]);
#endif
                if((!vcSportFlag) && (vcSpo2Value > 0))
                {
                    real_spo = vcSpo2Value;  血氧值
                }
```

图 - 6

注意

当模块能正常识别佩戴、摘下，也能正常调整亮度，但是无法输出心率值，vcHr02.vcFifoReadFlag 无法置 1，则是由于时钟校验没有通过，可进一步验证，在如下图-7 处添加 log，打印 ppgLength 的值，将手一直贴近模块，如果 ppgLength 一直为 0，则确实是时钟校验没有通过，如果 ppgLength 的值为 20，则需另行分析。

```
void vcHr02_process(AlgoSportMode_t vcSportMode)
{
    uint8_t algoCallNum = 0;
    uint8_t ppgLength = 0;
    uint8_t vcSportFlag = 0;

    if (vcHr02IRQFlag)
    {
        vcHr02IRQFlag = false;

        if(vcHr02.workMode == VCWORK_MODE_HRWORK)
        {
            if(VCHR02RET_UNWEARTOISWEAR == vcHr02GetSampleValues(&vcHr02,&ppgLength))
            {
                Algo_Init();
            }
              在本行添加日志，打印ppgLength的值
            if(vcHr02.vcFifoReadFlag || vcHr02.vcPsFlag)
            {
```

图 - 7

4. 外部中断处理流程

接收到中断事件后，调用 vcHr02GetSampleValues 获取当前的中断状态。根据获取到的中断状态做相应操作。示例操作详见下面流程图:

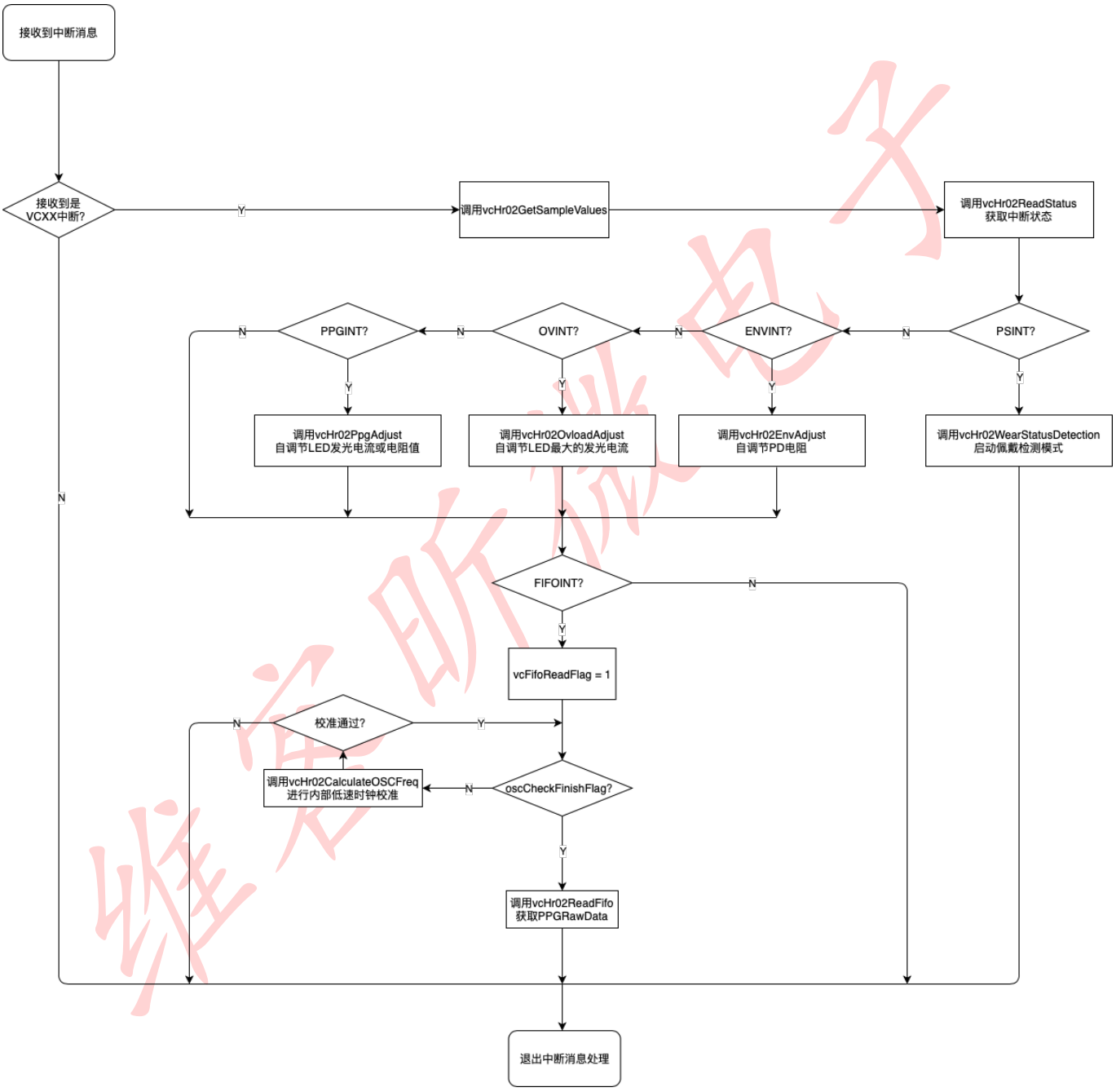

图 - 8

5. Gsensor 数据适配

运动心率，算法需要辅以 Gsensor 的数据进行计算，对于 Gsensor 的精度和量程等有以下要求：

1. Gsensor 采样率，不能低于 25Hz，至少为 25Hz；
2. Gsensor 的量程，不能低于±4G，可以高于±4G；
3. Gsensor 的精度，算法要求精度为±1G 8bit（不包含符号位，有效位为 8 位），所以当精度高于算法要求的精度时，需要降低精度传给算法。例如，客户 Gsensor 的量程精度为±8G 16bit，最高位为符号位（第 16bit），则有效位数为 15bit，即 8G 用 2^{15} 表示，1G 则用 2^{12} 表示，算法输入的 X、Y、Z 轴数据需右移 4 位；
4. Gsensor 的方向，算法要求 X 轴为垂直于手臂方向，Y 轴为沿着手臂方向；
5. Gsensor 数据获取方式，Gsensor 应带有 FIFO 缓存，获取 Gsensor 数据一般有中断触发获取和定时器定时读取两种方式，不同获取方式有不同的处理方式：
   1) 中断触发获取方式：Gsensor 中断触发后，将 Gsensor FIFO 里的数据缓存到数组中，下一次中断触发时更新成 Gsensor 的最新数据，当模块中断触发时，直接从数组中获取 X、Y、Z 的数据。注意，Gsensor 的中断频率不能低于模块的中断频率，模块 FIFO 中断的时间间隔为 800ms（1.25Hz，默认），则 Gsensor 的中断时间间隔应等于 800ms 或可以被 800ms 整除。当 Gsensor 与模块采样率相同时，直接将 Gsensor 数据缓存到数组中（数组大小可设置为 40），每次 Gsensor 中断更新 20 组数据，等待模块中断调用将最新 20 组数据传入算法；当 Gsensor 采样率高于模块采样率时，同样将 Gsensor 数组缓存于数组中，根据采样率抽取 20 组加速度数据（抽样数组，请联系 FAE）。详情请参照例程。
   2) 定时器定时读取方式：由于模块 FIFO 中断是稳定 800ms（默认），可以用模块 FIFO 中断时间间隔取代定时器定时时间，即直接在模块中断服务函数中相应位置读取 Gsensor 数据，当模块不工作时，再开启定时器获取 Gsensor 数据用作计步、抬腕亮屏等功能。如 Gsensor 支持定时读取 FIFO，可优先选用。Gsensor 采样率高于模块采样率（25Hz），800ms 模块会获取 20 组 ppg 数据，对应需要 20 组加速度数据，当 Gsensor 采样率高于模块采样率，则 Gsensor 每 800ms 获取到的数据会多于 20 个，这种情况需联系 FAE，告知 Gsensor 采样率，由 FAE 给出抽样数组。详情请参照例程。

算法对 XYZ 轴

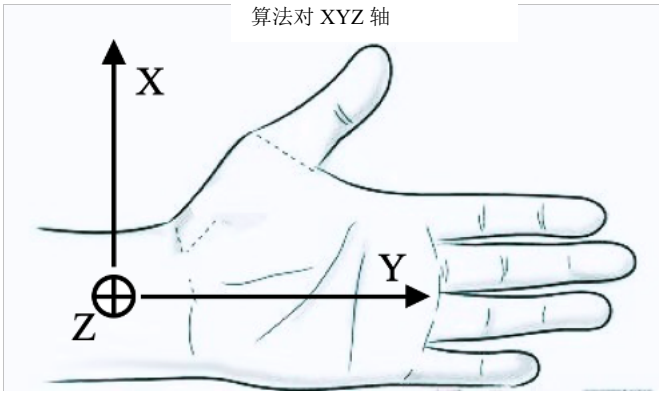

图 - 9

6. Example

适配 Gsensor 部分代码在中断服务函数 vcHr02_process(vcSportMode)中，以下将以心率模式适配 Gsensor 代码为例，血氧模式下适配与心率模式一致。

注：Gsensor 采样率为 36Hz，量程精度为±4G 16bit 以下为心率模式下部分代码

```
/*********************************************************************************************
*********************************************************************************************/
if (vcHr02.workMode == VCWORK_MODE_HRWORK)
{
    if (VCHR02RET_UNWEARTOISWEAR == vcHr02GetSampleValues(&vcHr02,&ppgLength))
    {
        Algo_Init();
    }
    if(vcHr02.vcFifoReadFlag || vcHr02.vcPsFlag)
    {
    /*Gsensor 数据适配部分开始*/
#if GsensorEn     //在 module_heart_vc_int.c 文件顶部将宏 GsensorEn 置 1

        ReadGsensorFIFO(xDat,yData,zData) ; //此处添加读取 Gsensor FIFO 数据的接口函数

        /* GsensorLength 为实际读到的三轴的数据组数，定时读取方式获得的数据组数可能多于 29 组，可能
           少于 29 组，但 800ms 读一次基本都会读到 29 组，根据客户实际情况修改
        */
        if(GsensorLength>=29)   //多于 29 组，则第 29 组之后都舍弃
        {
            GsensorLength = 29;
        }
        else if(GsensorLength>=1)
        {
            for(uint8_t i=GsensorLength;i<29;i++)   //少于 29 组，用最后一组数据将数据补齐至 29 组
            {
                xData[i]=xData[GsensorLength-1];
                yData[i]=yData[GsensorLength-1];
                zData[i]=zData[GsensorLength-1];
            }
        }

        for (uint8_t i=0;i<20;i++)
        {
           /* cash_num[20]数组为抽样数组；此处将 29 个数据抽成 20 个，匹配 800mS 中断的 20 个 PPG
              右移 5 位的操作是心率算法需要的三轴的数据是 256 代表 1G(1 个重力加速度)
           */
            xData[i] = yData[cash_num[i]]>>5;
            yData[i] = xData[cash_num[i]]>>5;
            zData[i] = zData[cash_num[i]]>>5;
        }
#endif
        if(vcHr02.vcFifoReadFlag)
        {
            vcHr02.vcFifoReadFlag = 0;
            if(vcHr02.wearStatus == VCHR02WEARST_ISWEAR)
            {
#if GsensorEn
                if (20 < ppgLength)                          /*当模块中断没有被及时响应时，下次读到的 ppg
                {                                              数据的组数会多于 20 个，则需要用最后一组的
                    for(uint8_t i = 29; i < ppgLength; i++)    三轴数据将三轴数据的总组数补到与 ppg 组数一
                    {                                          致，至此 Gsensor 数据适配完成*/
                        xData[i] = xData[29 - 1];
                        yData[i] = yData[29 - 1];
                        zData[i] = zData[29 - 1];
                    }
                }
#endif
```

```
                    algoInputData.envSample = vcHr02.sampleData.envValue[0];
                    for(algoCallNum= 0; algoCallNum < ppgLength; algoCallNum++)
                    {
                        algoInputData.ppgSample = vcHr02.sampleData.ppgValue[algoCallNum];
                        algoInputData.axes.x = xData[algoCallNum];//The direction vertical with ARM.
                        algoInputData.axes.y = yData[algoCallNum];//The direction parallel with ARM.
                        algoInputData.axes.z = zData[algoCallNum];//The direction upside.
                        Algo_Input(&algoInputData, 1000/vcHr02SampleRate, vcSportMode);
                    }

                    Algo_Output(&algoOutputData);
                    HeartRateValue = algoOutputData.hrData;

                    if(HeartRateValue == -1)
                    {
                        Algo_Init();
                    }
                }
            }
            else
            {
#if SportMotionEn
                green_led_off_state_gsensor_abs_sum_diff_func(xData[0],yData[0],zData[0]);
#endif
                HeartRateValue = -2;
            }
        }
}
```