GH30x 驱动库移植指南

V 3.1

2021-08-05

====== 免责声明======

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX 对因这些信息及使用这些信息而引起的后果不承担任何责任。未经GOODIX书面批准，不得将GOODIX 的产品用作生命维持系统中的关键组件。在GOODIX 知识产权保护下，不得暗中或以其他方式转让任何许可证。

目录

1 文档简介

注意

该文档适用的示例代码版本为：0.2.0

该文档适用的单驱动库版本为：0.7.0.2 及之后版本

1.1 本文档目的

本文档旨在帮助开发者利用 GH30x example 文件、驱动库文件快速实现 GH30x 芯片的正常使用。

1.2 驱动库框架

应用时可分为库文件以及 gh30x_process 等 example 代码。其中大致可以分为以下三层：

第一层：对接客户应用层接口。即下图中的 gh30x_process 层。

第二层：对接 GH30x 底层操作函数以及提供接口给第一层。即下图中的 gh30x_reg_array、gh30x_comm_pkg、gh30x_ctrl 层。

第三层：GH30x 底层操作函数。即下图中的 GH30x Port 层。

故 GH30x 驱动应用软件架构层级图如下图所示：

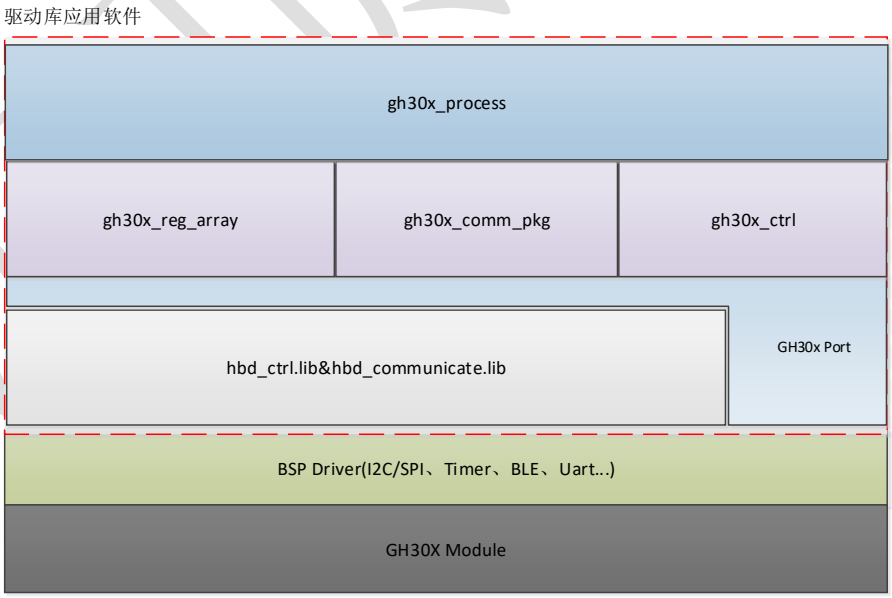

图 1-1 驱动层框架图

客户应用层需关注第一层和第三层，其中第一层代码是直接对接应用层的，整个调试过程中都需要关注，该层函数有一个对外的接口是 gh30x_example.h 文件和 gh30x_example_config.h 文件，gh30x_example.h 文件主要是给客户应用层调用的函数，gh30x_example_config.h 文件主要是设置支持的功能；第三层涉及到 I2C、GPIO 的底层操作，需要在前期调试时关注，该层函数体现在 gh30x_example_port.c 文件中，主要是 I2C、中断 GPIO、Timer 等的设置。

1.3 驱动流程框架

1、上电控制由平台硬件设计决定，模组初始化需要调用软件 init 接口。

上电及初始化流程如下图：

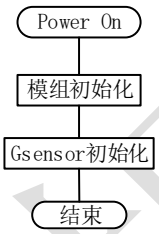

图 1-2 上电及初始化流程

2、在 G-sensor 中断中读取 G-sensor 数据，以及 G-sensor 移动判断。

完成初始化之后的 G-sensor 中断流程如下：

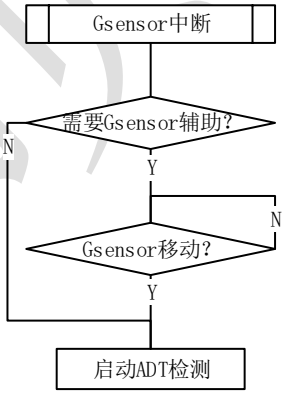

图 1-3 G-sensor 中断流程

3、在完成心率模组的初始化，调用对应的启动功能接口之后，心率模组的中断处理流程如下图：

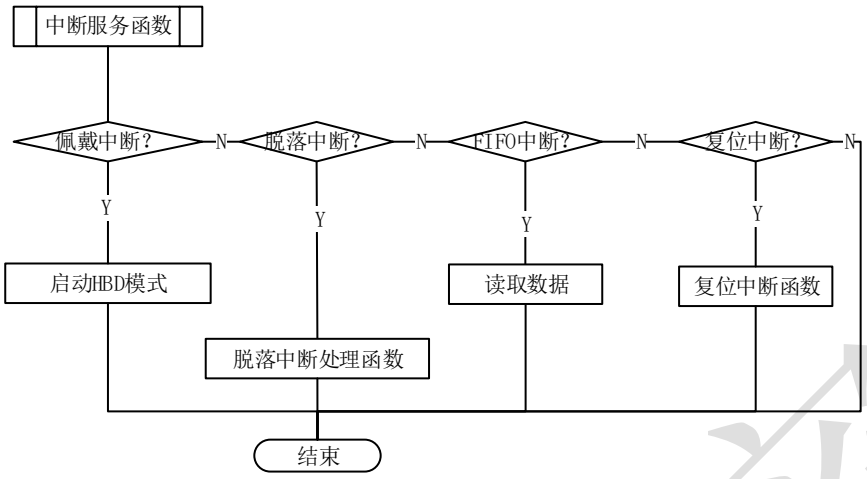

图 1-4 中断处理流程

1.4 支持的平台

已支持的平台有 ARM 的 M 系列和 A 系列，ARC，ESP32 等常见平台，需要了解是否有您需要的平台版本库请联系您的技术支持。

如果需要支持新的平台，需要提供对应平台的编译工具链、对应平台工程的编译选项、可编译通过的基础工程，如有可能尽量提供可调试硬件平台环境。

1.5 驱动应用文件介绍

gh30x_example.h：对外接口声明头文件，用于提供给客户上层应用调用；

gh30x_example_config.h：example 功能支持配置头文件；

gh30x_example_port.c：功能支持需要移植的接口，客户根据平台进行移植适配；

gh30x_example_reg_array.c：初始化配置以及不同模式的寄存器配置；

gh30x_example_common.h：example 通用宏和常量的定义，函数的声明等（客户无需关注）；

gh30x_example_comm_pkg.c：ble/uart 数据打包函数接口定义（客户无需关注）；

gh30x_example_ctrl.c：控制函数接口定义（客户无需关注）；

gh30x_example_process.c：应用流程控制函数定义（客户无需关注）；

cortexM4l_hbd_ctrl.lib：驱动算法库，必须包含，不同平台名称不同；

hbd_ctrl.h：驱动算法库对应的头文件，必须包含；

TestLib.lib：Local Test 产测库，视客户产测情况添加。

systemTest.h：Local Test 产测库对应的头文件，视客户产测情况添加。

1.5.1 gh30x_example_config.h 文件配置说明

表 1-1 gh30x_example_config.h 文件配置说明

__GH30X_COMMUNICATION_INTERFACE__	芯片 IIC/SPI 接口选择，接口为二选一，需实现 gh30x_example_port.c 文件中对应接口函数
__GH30X_IRQ_PLUSE_WIDTH_CONFIG__	芯片中断脉冲宽度配置为 255us 使能
__PLATFORM_DELAY_US_CONFIG__	注册平台延时 us 函数使能，需在 gh30x_example_port.c 实现平台 us 延时函数 hal_gh30x_delay_us
__RETRY_MAX_CNT_CONFIG__	启动失败重试次数
__RESET_REINIT_CNT_CONFIG__	初始化失败重试次数
__FIFO_INT_TIMEOUT_CHECK__	FIFO 中断 timeout 监控使能，客户平台可能会出现漏中断时必须使能，使能需实现 gh30x_example_port.c 中的相关接口
__GET_RAWDATA_BUF_LEN__	读取 FIFO 数据的数组长度，范围 1-256，使用栈空间
__USE_GOODIX_APP__	Goodix App 对接支持使能，若要上传 MCU 模式的 dbg 数据须打开__ALGO_CALC_WITH_DBG_DATA__
__NEW_DATA_MULTI_PKG_NUM__	BLE 单次发送多包数据数量配置，配置多包发送支持需要 BLE 支持长包发送
__BLE_PKG_SIZE_MAX__	BLE 发送最长包长度配置，一般对齐 BLE 中 MTU，但与手机有相关性，需确认手机支持情况
__BLE_MCU_PKG_BUFFER_MAX_LEN__	BLE 发送 MCU 模式数据 Buffer 长度
__UART_WITH_GOODIX_TOOLS__	UART 对应 Goodix 工具使能
__SYSTEM_TEST_SUPPORT__	Local mode 测试模式使能
__SYSTEM_TEST_DATA_CNT_CONFIG__	Local mode 测试数据点数配置
__EXAMPLE_DEBUG_LOG_LVL__	调试信息等级配置
__EXAMPLE_LOG_DEBUG_SUP_LEN__	调试信息最长支持配置

1.5.2 gh30x_example.h 文件对外函数说明

表 1-2 gh30x_example.h 文件对外函数说明

gh30x_module_init	GH30x 模块初始化，上电/重新上电必须调用
gh30x_module_config	GH30x 模块配置接口
gh30x_module_start	GH30x 模块启动接口
gh30x_module_start_without_adt	GH30x 模块启动函数，直接启动 HBD 模式，不经过 ADT 模式
gh30x_module_stop	GH30x 模块停止函数
ble_module_recv_data_via_gdcs	BLE 接收数据处理函数，在客户平台接收 Goodix App 对接 BLE 服务接收函数中调用
uart_module_recv_data	UART 接收处理函数，在客户平台接收 Goodix 工具 UART 数据接收函数中调用
gh30x_systemtest_start	启动产测功能

1.5.3 gh30x_example_port.c 文件客户实现函数说明：

1) GH30x 通信接口函数，IIC 与 SPI 可选其中一种，必须实现：

1.1) IIC：

hal_gh30x_i2c_init：客户平台 IIC 资源初始化函数；

hal_gh30x_i2c_write：客户平台 IIC 根据 Goodix 要求实现的写数据函数；

hal_gh30x_i2c_read：客户平台 IIC 根据 Goodix 要求实现的读数据函数；

1.2) SPI：

void hal_gh30x_spi_init：客户平台 SPI 资源初始化函数；

uint8_t hal_gh30x_spi_write：客户平台 SPI 根据 Goodix 要求实现的写数据函数；

uint8_t hal_gh30x_spi_read：客户平台 SPI 根据 Goodix 要求实现的读数据函数；

void hal_gh30x_spi_cs_set_low：客户平台 SPI 根据 Goodix 要求实现的 CS 引脚置低电平函数；

void hal_gh30x_spi_cs_set_high： 客户平台 SPI 根据 Goodix 要求实现的 CS 引脚置高电平函数；

2) 延时函数，必须实现平台精确的 us 级延时函数：

hal_gh30x_delay_us：客户平台 SPI 根据 Goodix 要求实现的 us 级延时函数；

3) 客户平台 IO 中断初始化函数，GH30x 中断必须实现，G-sensor 中断根据平台实现：

hal_gh30x_int_init：客户平台与 GH30x int 的 IO 设为外部中断初始化函数；

hal_gsensor_int1_init：客户平台与 G-sensor FIFO int 的 IO 设为外部中断初始化函数；

4) 客户应用处理算法结果与事件函数，必须实现，客户根据使能的模式实现：

handle_getrawdata_mode_result：客户应用获取 PPG 数据函数；

handle_wear_status_result：客户应用处理佩戴事件结果函数；

handle_system_test_result：客户应用处理量产测试结果函数；

5) BLE 通信函数：

ble_module_send_heartrate：通过标准心率服务发送心率值，对接普通心率 app 时实现；

ble_module_add_rr_interval：添加 RR 间期值到标准心率服务，对接普通心率 app 时实现；；

ble_module_send_data_via_gdcs：通过章节 3.5 中定义的服务发送数据，用于对接 Goodix 调试 APK，必须实现；

7) 定时器函数:

7.1) BLE 重复发送数据定时器，需要发送 MCU 数据时需实现，定时间隔视 ble 稳定性而定，推荐 50-100ms

ble_module_repeat_send_timer_start：BLE 重复发送数据定时器启动函数；

ble_module_repeat_send_timer_stop：BLE 重复发送数据定时器停止函数；

ble_module_repeat_send_timer_init：BLE 重复发送数据定时器初始化函数；

7.2) fifo 中断函数监控定时器，必须实现，定时间隔为 fifo 中断间隔加 80ms@25Hz、20ms@100Hz。

hal_gh30x_fifo_int_timeout_timer_start：fifo 中断函数监控定时器启动函数；

hal_gh30x_fifo_int_timeout_timer_stop：fifo 中断函数监控定时器停止函数；

hal_gh30x_fifo_int_timeout_timer_init：fifo 中断函数监控定时器初始化函数；

7) uart 通信函数，需对接 Goodix 量产工具/Dongle 工具时需实现：

uart_module_send_data：客户平台 uart 发送数据，用于对接 Goodix 工具；

8) 通信命令处理函数，需处理 goodix 工具命令时需实现：

handle_goodix_communicate_cmd：处理 Goodix 工具通信命令函数；

9) log 函数，必须实现：

example_dbg_log：客户平台输出 log 函数；

2 例程移植说明

以下说明中以 GH3011 为例，其它型号(如 GH300、GH301、GH3018 等)根据实际情况参考即可。

2.1 应用方案介绍

2.1.1 应用框架说明

GH3011 与 MCU 的通信方式可以选择 SPI 或 IIC 通讯；另外，GH3011 提供一路 INT 中断信号用于提示心率数据 Ready 和一些状态；在供电上， 需要外部提供 VLED 及 VCC 供电电源。

GH3011 有 3 个 LED 驱动电流通道，同一个驱动通道上可并联多个 LED，型号必须相同，驱动通道的选择需要满足表 2-1 所示要求。

表 2-1 GH3011 LED 电流驱动通道选择

驱动通道	“血氧+心率”应用		“心率”应用	
	LED 光源	分配功能	LED 光源	分配功能
LED_DRV0	红外光	佩戴检测、血氧、夜间心率检测	红外光	佩戴检测、夜间心率检测
LED_DRV1	红光	血氧	绿光	心率
LED_DRV2	绿光	心率	绿光	

GH3011 典型应用的系统框图如下图中的图 2-1 和图 2-2 所示。

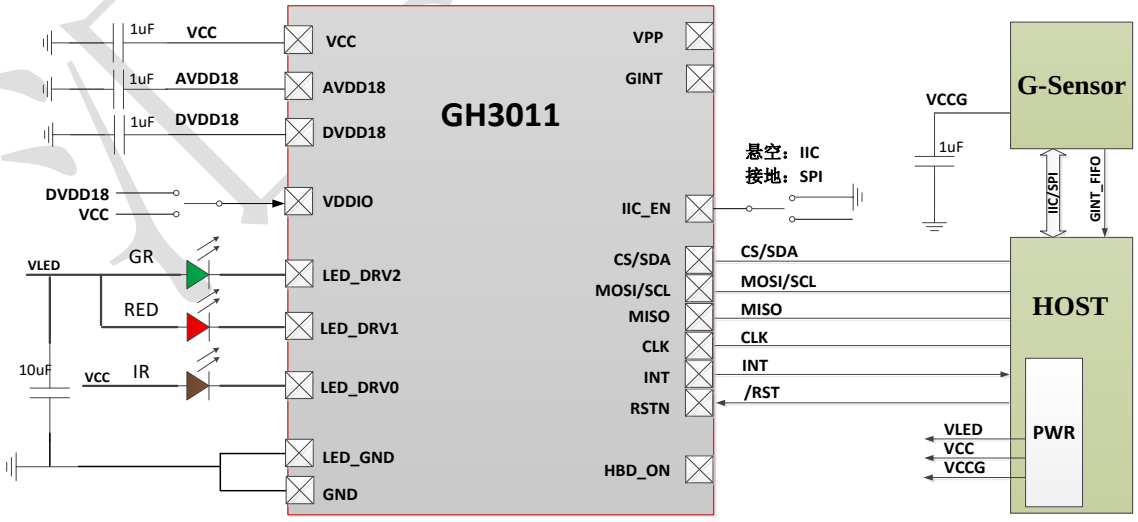

图 2-1 GH3011 典型心率及血氧检测应用方案系统方案框图

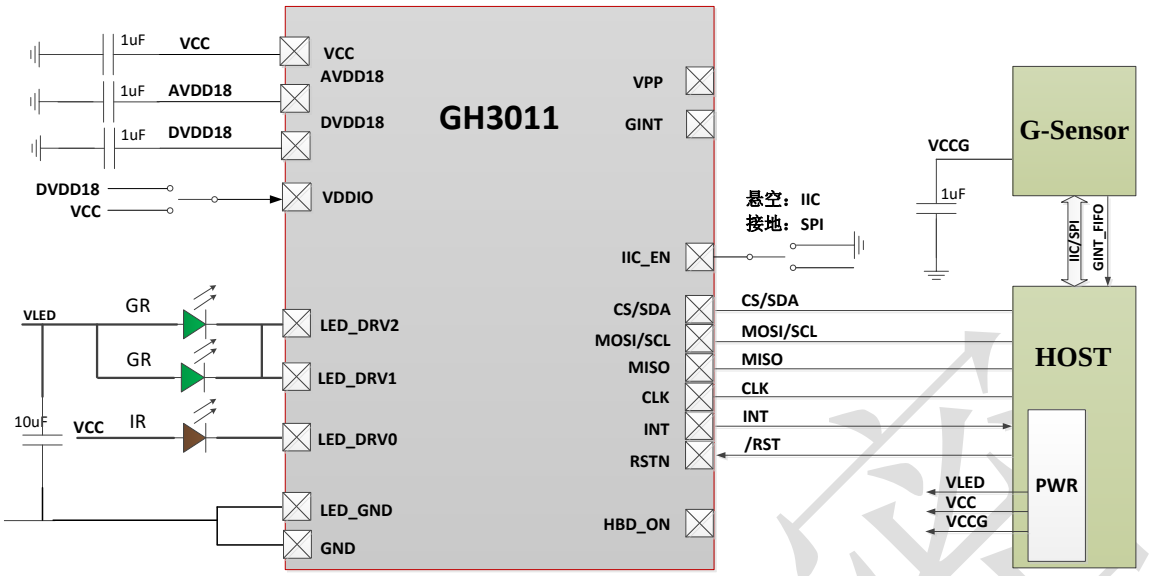

图 2-2 GH3011 典型心率检测应用方案系统框图（不支持血氧）

图 2-1 中支持心率、血氧、佩戴检测功能，图 2-2 中支持心率、佩戴检测功能。

2.1.1 佩戴检测

红外 ADT 检测：在调用 API 函数 gh30x_module_start 之后，红外 ADT 按照 5Hz 采样率检测红外是否被遮挡，如果有遮挡，则会产生佩戴中断，进入中断函数分支 INT_STATUS_WEAR_DETECTED，在佩戴中断函数 gh30x_wear_evt_handler 中，会调用 API 接口 gh30x_start_func_with_mode 启动 HBD 模式进行数据采集。

2.1.2 数据采集

心率模组在上电后需要调用 gh30x_module_init 进行心率模组，然后才可以启动 ADT 进行佩戴检测或启动 HBD 模式采集数据。

在调用 gh30x_module_start 或 gh30x_module_start_without_adt 启动采集前应调用 gh30x_module_config 进行配置。

进入 HBD 模式后会在中断函数 gh30x_fifo_evt_getrawdata_mode_only 中输出 PPG 数据，可以在函数 handle_getrawdata_mode_result 中获得并进行应用操作。

2.2 基本功能移植

2.2.1 驱动库移植到工程

首先需要将驱动库移植到工程中。

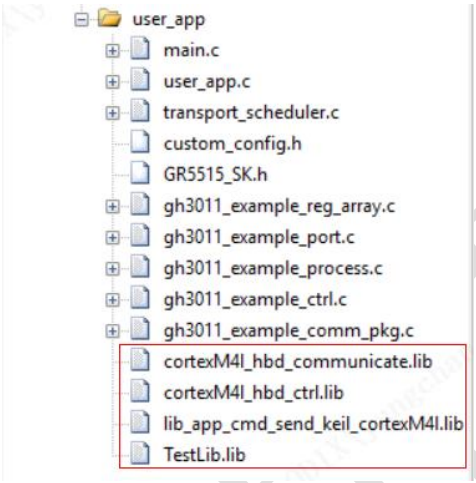

图 2-3 驱动库移植到 keil 工程图

将 cortexM4l_hbd_ctrl.lib、TestLib.lib(视客户产测情况)以及库对应的头文件添加到工程中，具体操作方法因编译软件区别可能有差异。

2.2.2 通信功能 Bring Up

GH30x 上电时序图如下：

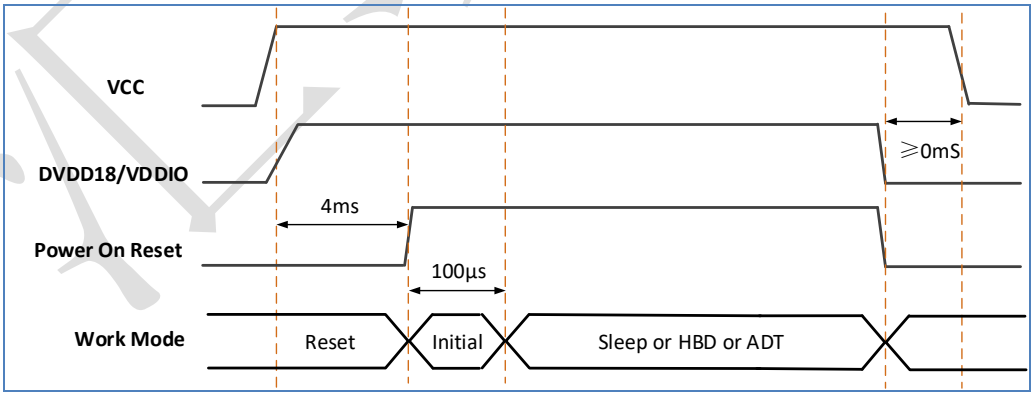

图 2-4 上电时序图

该时序图较明确的规定了上电后需要完成的相关初始化工作及精准的 us 级延时，所以首先需要根据平台实现精准的 us 级延时函数。在 gh30x_example_port.c 文件中需要实现 hal_gh30x_delay_us 函数，根据实际的应用平台实现精准的 us 级延时函数，具体实现方法因平台差异可能稍有不同。

延时函数完成后接下来开始设置 I2C 通信。I2C 通信首先是 I2C 引脚的设置，一般会进行如下的设置：将引脚设置为 I2C 引脚，设置内部上拉(若外部硬件增加了上拉可不用内部上拉)，设置 I2C 通信速率，设置 I2C 数据传输方式等。

设置好 I2C 引脚后开始编写 I2C 的读写函数，因为对 GH30x 的所有操作都是通过这两个 I2C 读写函数实现的，所以要保证 I2C 读写函数的准确性。在 gh30x_example_port.c 文件中需要实现如下的函数：hal_gh30x_i2c_write 接口函数和 hal_gh30x_i2c_read 接口函数。需要注意的是 I2C 的读函数需要先利用 I2C 写入一个数据再进行读的操作。

I2C 读写函数编写完成后就需要验证 I2C 是否能正常工作。在应用工程中调用 gh30x_module_init 函数接口并利用 UART 的打印信息来判断 I2C 是否正常。如果 I2C 读写函数不正常或者 I2C 引脚设置不正确 UART 都会出现打印类似于“gh30x init error[*]”的 log 信息，如果 I2C 读写正常并且 I2C 引脚设置正确 UART 都会出现类似于“gh30x module init ok”的 log 信息。

2.2.3 配置文件修改

如果是严格按照汇顶推荐光路规范的结构不需要修改配置文件，如果不是则需要修改配置文件，需要修改的配置文件在 gh30x_example_reg_array.c 文件中。使用 EVK 工具生成配置文件，然后将配置文件中的数组内容替换到 gh30x_example_reg_array.c 文件中相关的数组中。
gh30x_example_reg_array.c 文件中需要替换的有如下配置数组：hb_adt_confirm_reg_config、hb_reg_config_array、spo2_reg_config_array。

通过 EVK 工具生成的文件如下图：

图 2-5 EVK 工具生成的配置文件图

其中 EVK 生成的配置文件与驱动配置对应关系表如下图：

表 2-2 EVK 生成的配置文件与驱动配置对应关系表

GH3011_High_auto_ADT_BO_xxxx	(未使用)

GH3011_High_auto_ADT_HB_xxxx	hb_adt_confirm_reg_config
GH3011_High_auto_BO_xxxx	spo2_reg_config_array
GH3011_High_auto_HB_xxxx	hb_reg_config_array

需要将文件中的数组全部替换驱动中对应的配置数组。

2.2.4 用户实现模块

后续需要实现的模块如下：INT 中断、BLE 数据透传、Local Test 产测(选用)等功能。

首先是 INT 中断。INT 中断首先需要设置 INT 引脚为中断引脚。其次在中断处理函数中增加函数进行处理，也就是在 INT 引脚的中断处理函数中使用 gh30x_int_msg_handler 函数进行 INT 引脚的中断处理，此时就可以通过算法计算输出心率值了。

其次是 BLE 数据透传。BLE 透传需要完成两部分的工作：添加服务和添加 Timer 定时器。其中服务添加后需要使用到的最重要的是两个函数是 **gh30x_app_cmd_parse** 函数和 **ble_module_send_data_via_gdcs** 函数，其中 gh30x_app_cmd_parse 函数是处理 BLE 接收到的数据，ble_module_send_data_via_gdcs 函数是将心率相关的数据通过 BLE 发送到手机。

该步骤需要新建一个服务，与 GOODIX APP 对接的 BLE profile 需要按照以下设定：

服务 UUID："0000190e-0000-1000-8000-00805f9b34fb"

数据 Tx characteristic，连接成功后等待接收 BLE 设备端数据

TX_UUID："00000003-0000-1000-8000-00805f9b34fb"

数据 Rx characteristic 属性 UUID，连接成功后可发送数据到 BLE 设备端

RX_UUID："00000004-0000-1000-8000-00805f9b34fb"

新建服务完成之后在该服务的接收中断处理事件中调用如下函数：gh30x_app_cmd_parse 函数，并且需要在 gh30x_example_port.c 文件中的 ble_module_send_data_via_gdcs 函数里调用该服务的数据发送函数。除此之外，还需事先 BLE 重复发送数据定时器，需要发送 MCU 数据时需实现，定时间隔视 ble 稳定性而定，推荐 50-100ms，且需实现的 gh30x_example_port.c 文件中的 Timer 函数如下：

ble_module_repeat_send_timer_start：BLE 重复发送数据定时器启动函数；

ble_module_repeat_send_timer_stop：BLE 重复发送数据定时器停止函数；

ble_module_repeat_send_timer_init：BLE 重复发送数据定时器初始化函数；

至此 BLE 透传基本移植完成。

接下来根据客户情况进行 Local Test 产测功能添加。具体可参考 2.3.3.1 节。

2.3 重要模块应用说明

2.3.1 蓝牙透传调试说明

具有蓝牙功能的项目应用可以通过蓝牙连接手机 APP，将算法库相关数据通过无线传输到手机，为分析问题和性能验收测试提供可靠的渠道。蓝牙传输必须使用低功耗蓝牙（BLE, Bluetooth Low Energy）进行数据传输，蓝牙 4.0 及以上版本均可。下面介绍实现蓝牙透传调试需要进行的操作。

2.3.1.1 新增蓝牙服务

在蓝牙主控芯片 BLE GATT 初始化代码中增加一个自定义服务和两个特征，要求如下：

自定义服务 UUID："0000190e-0000-1000-8000-00805f9b34fb"；

- 说明：GH30x 相关 APP 专用 BLE 服务，用于算法相关数据通讯使用

发送特征 UUID："00000003-0000-1000-8000-00805f9b34fb"；

- 说明：GH30x 相关 APP 专用 BLE 特征，用于设备向手机上传算法相关数据使用。

接收特征 UUID："00000004-0000-1000-8000-00805f9b34fb"

- 说明：GH30x 相关 APP 专用 BLE 特征，用于手机向设备下发算法相关指令使用。

各蓝牙主控平台均可实现上述增加自定义服务和特征操作，如下宏定义可以直接复制到程序中使用，该步骤完成后可以使用任意 BLE 调试 APP 连接设备验收服务添加是否完成，本文以汇顶 BLE 调试工具 GRToolbox APP 为例服务初始化正常如下图所示。

```
#define GH30X_SERVICE_UUID {0xFB, 0x34, 0x9B, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0x0E, 0x19, 0x00, 0x00}

#define GH30X_TX_UUID {0xFB, 0x34, 0x9B, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00}

#define GH30X_RX_UUID {0xFB, 0x34, 0x9B, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00}
```

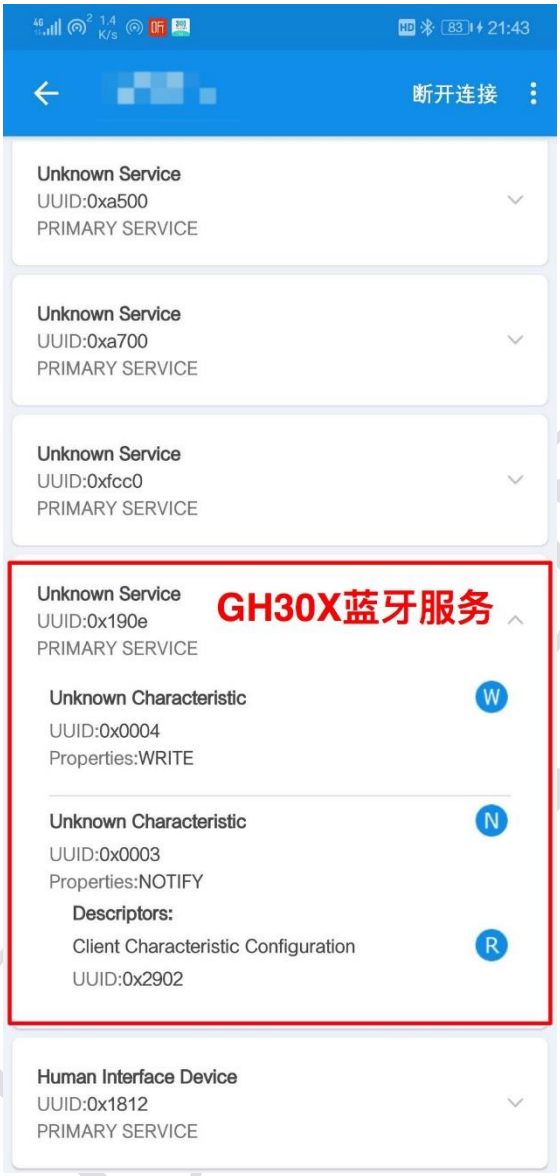

图 2-6 服务初始化界面

2.3.1.2 注册蓝牙接收数据回调函数

GH30X 相关 APP 在连接设备后会寻找所需蓝牙服务和特征，当找到后会通过设备接收特征下发指令，此时 BLE 主控软件会触发通知事件，用户代码需要调用示例代码中提供的蓝牙数据解析函数，并且将蓝牙接收数据的指针和长度传递到函数中，如下函数在示例代码 gh30x_example_port.c 文件中实现，需要确保此文件加入工程参与编译；该函数在 gh30x_example_common.h 文件中声明，需要确保此文件加入工程头文件搜索目录。

```
ble_module_recv_data_via_gdcs(data, length);
```

2.3.1.3 注册蓝牙发送数据回调函数

设备在收到 GH30x 相关 APP 发送的数据后需要进行回复，此时算法库会自动调用已经注册好的蓝牙发送函数进行数据上传操作，示例代码在 GH30x 算法库初始化阶段调用

gh30x_comm_pkg_init()函数将 BLE 发送函数注册到算法库中，注册的回调函数在需要发送数据时调用如下函数，用户需要在该函数实现中增加其蓝牙主控平台发送数据函数调用（本文以红色语句为例，用户需替换红色语句），该函数实现在 gh30x_example_port.c 文件中，附带两个参数，分别是发送数据的指针和长度。

```
uint8_t ble_module_send_data_via_gdcs(uint8_t data[], uint8_t length)
{
  uint8_t ret = GH30X_EXAMPLE_OK_VAL;
  // code implement by user
  ble_tx_data_send(data, length);
  return ret;
}
```

2.3.1.4 注册 MCU 模式发送数据定时器相关函数

蓝牙 MCU 模式需要使用一个定时器作为事件驱动发送数据，GH30x 算法库初始化阶段调用 gh30x_comm_pkg_init()函数将定时器初始化，用户需要在下面的函数中实现定时器初始化，该函数实现在 gh30x_example_port.c 文件中，定时器建议配置为 100 毫秒触发，且蓝牙连接间隔需要小于定时器触发时间 100 毫秒。

```
void ble_module_repeat_send_timer_init(void)
{
      // code implement by user
      // must register func ble_module_repeat_send_timer_handler as callback
  /* should setup 100ms timer and ble connect interval should < 100ms
   */
}
```

同时需要注册开启定时器、关闭定时器两个函数实现，并且在定时器触发函数中增加发送数据函数调用，如下。

```
void ble_module_repeat_send_timer_start(void)
{
      // code implement by user
}

void ble_module_repeat_send_timer_stop(void)
{
      // code implement by user
}

void TIMER_IRQHandler(void)
{
      ble_module_repeat_send_timer_handler();
}
```

2.3.1.5 验收蓝牙透传功能

按照上述步骤完成蓝牙透传调试后可以使用任意调试 APP 进行功能验收，

1、将设备正常佩戴在合适位置，使用任意 BLE 调试 APP 连接设备；

2、打开发送特征通知功能，APP 界面如下图；

3、使用接收特征写入四字节十六进制数据 0xC022452C，此时设备会开启绿灯，可以看到绿灯在闪烁，APP 界面如下图；

4、发送特征先收到一次四字节十六进制数据 0xC022452C，随后持续收到数据，表示透传功能已正常工作，APP 界面如下图。

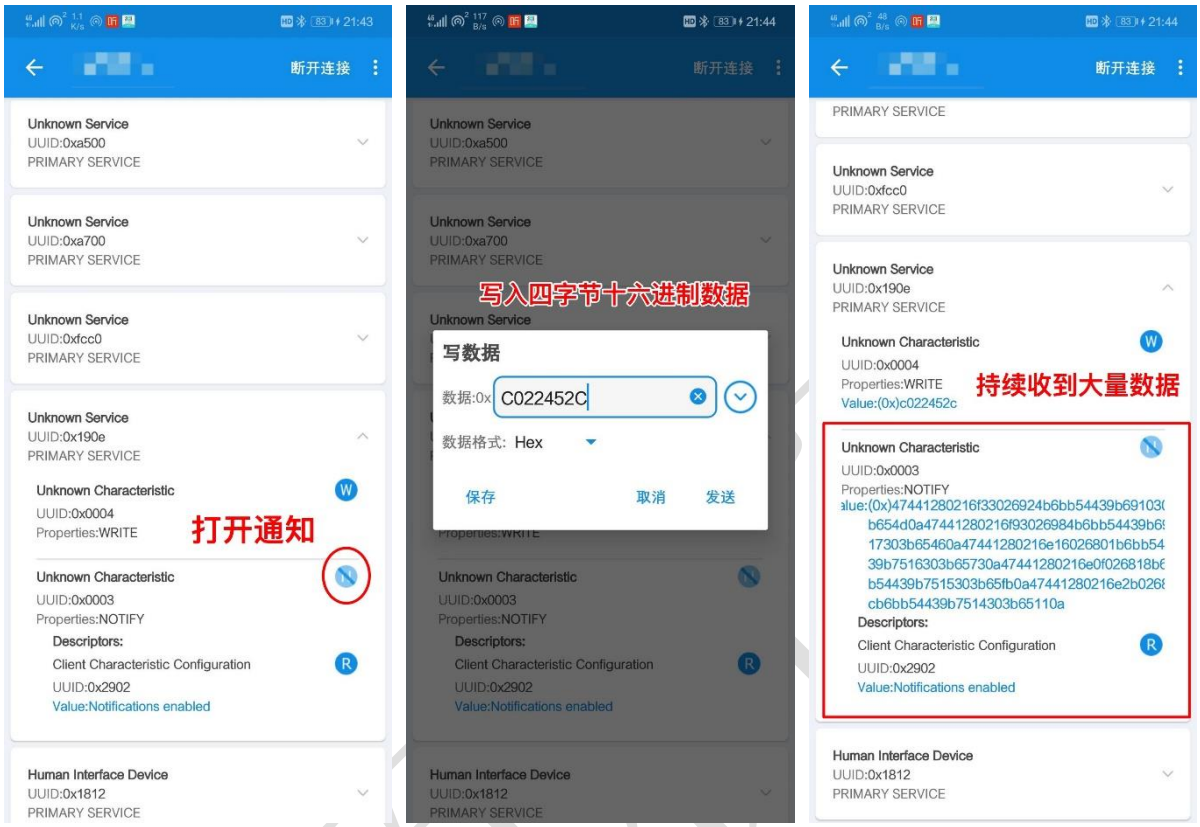

图 2-7 透传功能界面

蓝牙透传调试包含两种特点不同的应用模式，下面详细介绍两种模式的应用场景和区别：

- MCU 模式

  MCU 模式使用设备固件中的算法库执行运算，在算法执行完毕后将本次运算的结果和参与运算的 PPG、ACC 源数据打包通过蓝牙发送至手机，通常每秒开启一轮蓝牙发送，需要在下次

蓝牙发送开启前将所有数据发送完毕；心率模式每秒发送 650 字节左右，血氧模式每秒发送 2600 字节左右；该模式需要配合 2.2.4 章节初始化的定时器一起使用。

特点：APP 仅作为数据接收者不会对设备固件运行产生影响，可以最大限度做到与终端用户使用效果一致。

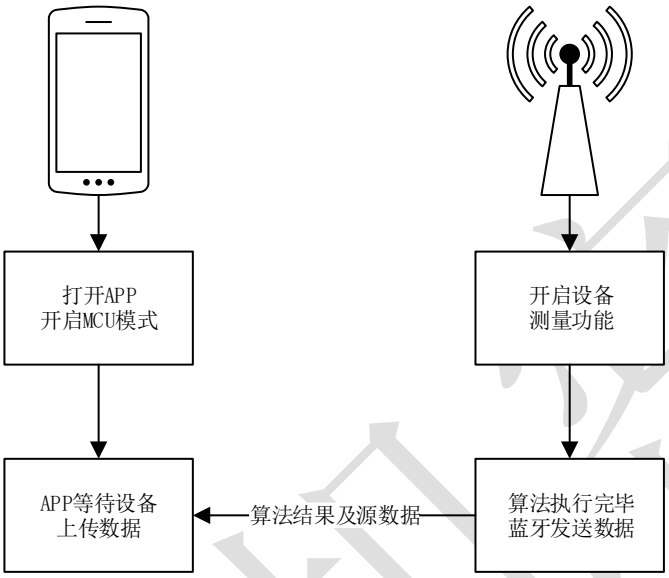

图 2-8 蓝牙透传调试应用模式

- APP 模式

  APP 模式使用手机 APP 中的算法库执行运算，设备在每次采样结束后通过蓝牙将数据上传至手机，每次发送的数据长度不会超过 20 字节，但发送间隔较 MCU 模式频繁，心率模式每 40ms 发送一次数据，血氧模式每 10ms 发送一次数据，需要考虑手机蓝牙对于连接间隔的承受能力。

  特点：算法库集成在 APP 中，可以通过升级 APP 版本实现升级算法库版本；APP 启动测量时下发配置，可以通过修改 APP 中的配置快速验证配置修改后的结果；避免对机器进行整个固件的空中升级操作，节约时间。

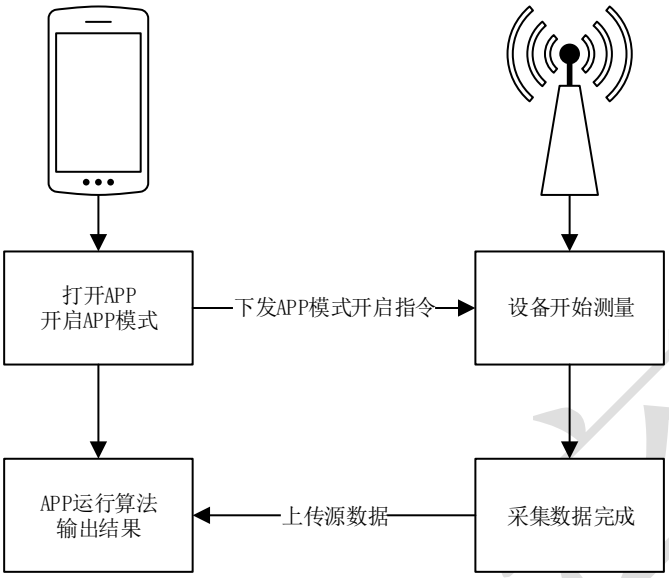

图 2-9 App 模式

2.3.1.6 设置与蓝牙透传功能相关的宏定义

__USE_GOODIX_APP__：使用蓝牙透传调试功能连接 GH30x 相关 APP 需开启此宏定义，可以在调试版本打开此宏定义，在发布版本关闭此宏定义。

__ALGO_CALC_WITH_DBG_DATA__：使用蓝牙透传 MCU 模式需开启此宏定义，与__USE_GOODIX_APP__宏定义配套使用。

__NEW_DATA_MULTI_PKG_NUM__：使用蓝牙透传功能在一次调用数据发送函数时拼包发送多个数据包，该宏定义数量加 1 为拼包数量，默认值 4，代表每次拼 5 包数据，共计 5*22=110 字节。

__BLE_PKG_SIZE_MAX__：使用蓝牙发送最长数据包定义，一般对齐蓝牙连接后更新的 MTU，该定义与手机蓝牙强相关，需确认手机支持情况合理设置。

__BLE_MCU_PKG_BUFFER_MAX_LEN__：使用蓝牙透传 MCU 模式数据缓冲区长度，根据最小计算公式：(DBG_MCU_PKG_RAW_FRAME_LEN * __ALGO_CALC_DBG_BUFFER_LEN__) + MCU_PKG_SPO2_ALGO_RESULT_LEN 得出结果为 2683 字节。

DBG_MCU_MODE_PKG_LEN：蓝牙 MCU 模式数据包个数，保持默认不修改。

DBG_MCU_PKG_RAW_FRAME_LEN：蓝牙 MCU 模式数据帧长度，保持默认不修改。

DBG_MCU_PKG_HEADER_LEN：蓝牙 MCU 模式数据包头长度，保持默认不修改。

注意事项：

1、使用蓝牙透传调试功能时请确认蓝牙连接间隔（Connection Interval）设置是否可以满足数据传输使用，APP 模式心率应用要求连接间隔小于 40 毫秒，血氧应用要求连接间隔小于 10 毫秒；MCU 模式心率应用要求每秒可以发送 650 字节，MCU 血氧应用要求每秒可以发送 2600 字节。

2、使用蓝牙 MCU 模式需要确认蓝牙 MTU 长度，避免应用层单次发送的数据长度大于蓝牙最大单次发送的数据长度造成堵塞蓝牙芯片正常工作。

2.3.2 工程模式测试调试说明

2.3.2.1 MCU 本地测试

对于产测而言，一般是 MT 测试(或者整机测试)和 MCU 本地测试二选一，即 MCU 本地产测是非必须的，需要视客户情况而定。

如果客户确定使用 MCU 本地测试，需要进行如下准备：GH30x 产测库、GH30x 模组、乳白色硅胶头、黑色吸光硅胶头、参考《GH30x 操作库移植指南》。

MCU 本地产测有几个使能开关和测试的接口：

__SYSTEM_TEST_SUPPORT__： ***Local mode*** 测试模式使能；

gh30x_module_system_test_otp_check： 系统测试 ***otp*** 函数；

gh30x_module_system_test_os_start： 系统测试 ***os*** 函数；

一般测试包括放置白色测试头的光路灵敏度测试和放置黑色吸光硅胶头的漏光测试，由于漏光和灵敏度测试的物理要求不一样，所以可以利用物理按键来触发漏光和灵敏度测试事件。

至此准备工作就完成了，下一步是开始测试。可以在调用 main 函数之前调用以下两个函数：gh30x_module_system_test_otp_check 函数和 gh30x_module_system_test_os_start 函数。

gh30x_module_system_test_otp_check 函数打印值如下表所示：

表 2-3 gh30x_module_system_test_otp_check 函数打印值

gh30x_module_system_test_otp_check 函数打印值	测试结果
0	ok
1	err
2	parma err

gh30x_module_system_test_os_start 函数打印值如下表所示：

表 2-4 gh30x_module_system_test_os_start 函数打印值

gh30x_module_system_test_os_start 函数打印值	测试结果
0	ok
1	rawdata err
2	noise err
3	para err

测试启动后根据得到的如下接口的结果需要基于最后一个值来获取 10pcs 的数据得到平均值再上下取 40%的余量设置为对应灯的上下限，替换 99999 和 37，三个灯都需要调整。

```
ROMAHBData systemtest_led0_os_result =
{
    34, 99999, 37, 0,
};
```

调整完成之后 MCU 本地测试就完成了。

2.3.2.2 UART 测试

FPM02 工具板上预留了 UART 测试接口，该 UART 接口定义如下图 8 所示：

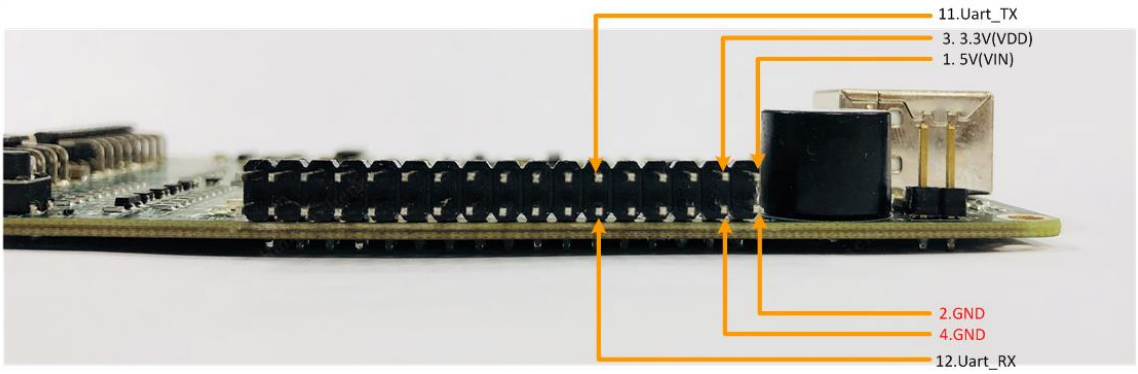

图 2-10 UART 通讯接口(J6)定义(侧视图)

其中相关引脚描述如下图 9 所示：

表 2-5 UART 通讯接口(J6)引脚描述

管脚序号	管脚名称	描述
1	5V（VIN）	5V 供电电源，根据测试设备需求选择使用
2	GND	连接模组的 GND 网络
3	3.3V（VDD）	3.3V 供电电源，根据测试设备需求选择使用
4	GND	连接模组的 GND 网络
11	I2C1_SDA	复用 UART Tx，连接模组 UART_TX 网络
12	I2C1_SCL	复用 UART_Rx，连接模组 UART_RX 网络

当使用 UART 测试时，需要按照实际情况实现 gh30x_example_port.c 文件中的 uart_module_recv_data 函数和 uart_module_send_data 函数。其中除了这两个函数之外还需要实现整个 UART 的功能，包括引脚设置，UART 功能设置等。

uart_module_recv_data 函数主要用于从 UART 接收相关控制命令用于返回数据及控制 GH30x 功能，uart_module_send_data 函数主要用于发送相关的数据给外部便于分析。

2.3.3 漏中断处理方案

由于平台可能会出现漏中断的情况，所以添加了漏中断的处理方案。

1、 首先，我们需要开启位于“gh30x_example_config.h”中有关于 FIFO 中断 timeout 监控使能的宏定义：__FIFO_INT_TIMEOUT_CHECK__；当置 1 时开启 FIFO 漏中断使能机制；置 0 时则不开启。

2、 其次，我们需要实现“gh30x_example_port.c”中以下三个函数（必须实现）：fifo 中断函数监控定时器，定时间隔为 fifo 中断间隔加 80ms@25Hz、20ms@100Hz。

hal_gh30x_fifo_int_timeout_timer_start： fifo 中断函数监控定时器启动函数；
该函数主要的目的就是实现定时器中断使能开启。

hal_gh30x_fifo_int_timeout_timer_stop： fifo 中断函数监控定时器停止函数；
该函数主要的目的就是实现定时器中断使能关闭。

hal_gh30x_fifo_int_timeout_timer_init： fifo 中断函数监控定时器初始化函数；
该函数主要是实现定时器的初始化工作，包括定时器模式、定时时长、定时器中断处理事件等设置。

实现要点：

客户只需关注以下几点即可：

1) 根据需求在“gh30x_example_config.h”文件中使能__FIFO_INT_TIMEOUT_CHECK__；
2) 根据平台资源实现对应函数：

hal_gh30x_fifo_int_timeout_timer_init;

hal_gh30x_fifo_int_timeout_timer_start;

hal_gh30x_fifo_int_timeout_timer_stop;

以下流程图是有关 FIFO 漏中断定时器相关的程序流程以及漏中断之后的处理。
TIMER_INIT: hal_gh30x_fifo_int_timeout_timer_init;
TIMER_START: hal_gh30x_fifo_int_timeout_timer_start;
TIMER_STOP: hal_gh30x_fifo_int_timeout_timer_stop;

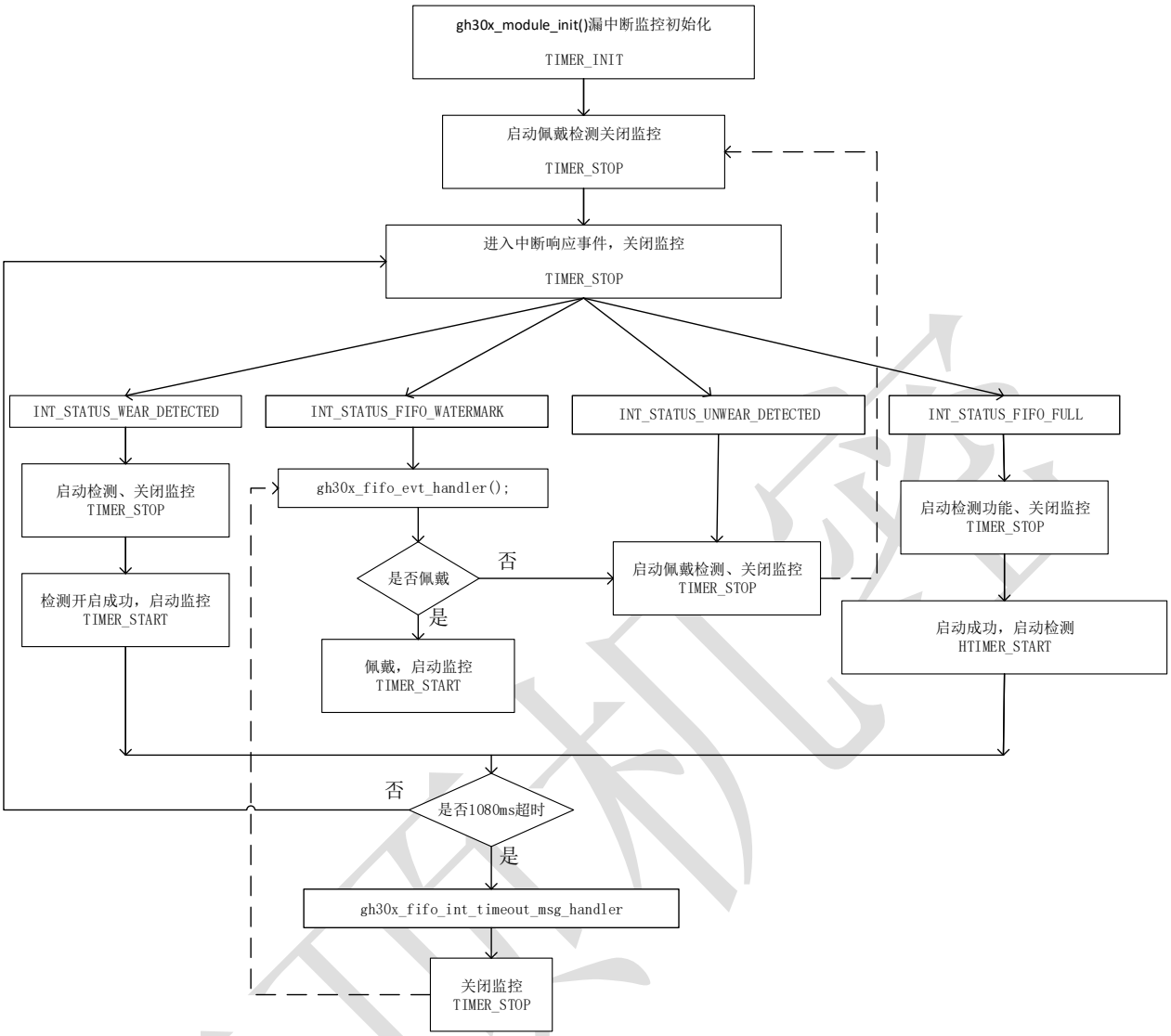

图 2-11 FIFO 漏中断定时器相关的程序流程

3 驱动库 API 说明

本说明基于 v0.5.3.2 版本驱动库。

表 3-1 驱动库 API 说明

函数名称	HBD_SetI2cRW
用途	注册 I2C/SPI 读写函数
入参	uchI2cIdLowTwoBitsSelect 地址设置，使用 EM_HBD_I2C_ID_SEL 枚举体定义即可 *pI2cWriteFunc 写函数 *pI2cReadFunc 读函数
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_PARAMETER_ERROR 入参异常

函数名称	HBD_SetDelayUsCallback
用途	注册微秒延时函数
入参	*pDelayUsFunc　　延时函数
出参	无
返回值	无

函数名称	HBD_LoadNewRegConfigArr
用途	加载配置表，通常在启动测试前调用
入参	StRegConfigArr[]　　配置表数组 usRegConfigLen　　数组长度(非字节长度)
出参	无
返回值	HBD_RET_OK 加载成功 HBD_RET_COMM_ERROR 加载失败

函数名称	HBD_CommunicationInterfaceConfirm
用途	检查 IIC/SPI 通信接口是否正常
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_COMM_ERROR 芯片通讯异常 HBD_RET_COMM_NOT_REGISTERED_ERROR 未注册芯片通讯接口

函数名称	HBD_SimpleInit
用途	驱动初始化
入参	*stHbdInitConfigParam　　初始化 ADT 配置
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_PARAMETER_ERROR 参数异常 HBD_RET_COMM_ERROR 芯片通讯异常 HBD_RET_COMM_NOT_REGISTERED_ERROR 未注册芯片通讯接口
函数名称	HBD_HbDetectStart
用途	启动心率测试
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_LED_CONFIG_ALL_OFF_ERROR 配置异常，没有开 LED HBD_RET_NO_INITED_ERROR 未初始化，需要调用 HBD_SimpleInit

问题反馈

函数名称	HBD_Stop
用途	停止运行，芯片进入休眠
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_NO_INITED_ERROR 未初始化，需要调用 HBD_SimpleInit

函数名称	HBD_AdtWearDetectStart
用途	启动一次 ADT 佩戴检测
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_NO_INITED_ERROR 未初始化，需要调用 HBD_SimpleInit

函数名称	HBD_AdtWearContinuousDetectStart
用途	启动持续的 ADT 佩戴检测，即在中断产生后会继续进行 ADT 检测，检测模式(佩戴检测或脱落检测)会自动翻转
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_NO_INITED_ERROR 未初始化，需要调用 HBD_SimpleInit

函数名称	HBD_IsAdtWearDetectHasStarted
用途	检查是否已启动 ADT 佩戴检测
入参	无
出参	无
返回值	0 未启动 1 已启动

函数名称	HBD_GetIntStatus
用途	获取中断类型
入参	无
出参	无
返回值	0 芯片复位中断 1 新数据就绪中断 2 达到 FIFO 阈值中断 3 满 FIFO 中断 4 佩戴中断

问题反馈

	5 脱落中断 6 无效中断

函数名称	HBD_GetFifoCntHasRead
用途	获取上次读取的 FIFO 数据帧数
入参	无
出参	无
返回值	上次读取的 FIFO 数据帧数

函数名称	HBD_ChipReset
用途	复位芯片
入参	无
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_COMM_NOT_REGISTERED_ERROR 未注册芯片通讯接口

函数名称	HBD_SetIrqPluseWidth
用途	设置中断脉冲宽度，下次启动测试生效
入参	uchIrqPluseWidth 脉冲宽度，不能为 0
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_PARAMETER_ERROR 参数错误

函数名称	*HBD_GetHbdVersion
用途	获取驱动库版本号字符串
入参	无
出参	无
返回值	版本号字符串

函数名称	HBD_StartHBDOnly
用途	纯采集功能启动
入参	usSampleRate 采样率，若设置为 0 则采样率直接由寄存器控制 ucEnableFifo 1 使能 FIFO，0 禁用 FIFO ucFifoTHR FIFO 中断阈值
出参	无
返回值	HBD_RET_OK 成功 HBD_RET_LED_CONFIG_ALL_OFF_ERROR 配置异常，没有开 LED HBD_RET_NO_INITED_ERROR 未初始化，需要调用 HBD_SimpleInit

问题反馈

函数名称	HBD_GetRawdataByFifoInt
用途	用于 FIFO 中断处理的获取纯采集功能的 FIFO 数据
入参	ucBufLen　　保存数据的 buf 帧长度，1 帧为两个 PPG 数据 nRawdataOut[]　保存数据的 buf，数组大小应为 unBufLen*2 *pucRealDataLen　实际输出的数据帧数
出参	无
返回值	0 获取成功 1 软件调光失败

函数名称	HBD_GetRawdataByNewDataInt
用途	用于新数据就绪中断处理的获取纯采集功能的数据
入参	*ppg1　PPG1 数据 *ppg2　PPG2 数据
出参	无
返回值	0 获取成功 1 软件调光失败

函数名称	HBD_GetRawdataHasDone
用途	读取完数据后调用，使芯片恢复低功耗 HBD 模式
入参	无
出参	无
返回值	无

4 移植常见问题及注意事项说明

- **移植 example 需要关注哪里文件？**

客户只需关注：

gh30x_example.h：对外接口声明头文件， 用于提供给客户上层应用调用；

gh30x_example_config.h： example 功能支持配置头文件；

gh30x_example_port.c：功能支持需要移植的接口，客户根据平台进行移植适配；

gh30x_example_reg_array.c：初始化配置以及不同模式的寄存器配置；

- **更换寄存器数组在哪个文件？对应模式的寄存器数组是哪些？**

问题反馈

gh30x_example_reg_array.c 文件；

hb_adt_confirm_reg_config：心率 adt 检测寄存器数组；

hb_reg_config_array：心率检测寄存器数组；

spo2_reg_config_array：血氧检测寄存器数组；

需要进行寄存器数组更改时，目前只需关注以上数组的更改，常见的修改频率的寄存器地址为：0x0016：对应的频率为：0x51e(25HZ)，0x0147(100HZ)。

- **算法库接口吐值函数对于 Gsensor 的要求？**

Gsensor 的传入量一定需要大于 rawdata 的值，不然心率计算接口等会吐出 0 值；确保 Gsensor 的频率大于心率 IC 的打码频率。

- **怎么配置心率算法场景？**

调用 gh30x_example.h 中 GH30X_HBA_SCENARIO_CONFIG，需在启动心率模式前调用。

使用见示例代码：GH30X_HBA_SCENARIO_CONFIG(2);

gh30x_module_start(GH30X_RUN_MODE_HB);

心率模式可以设置场景以提高准确性， 每次在 gh30x_module_start 前调用

设置 GH30X_HBA_SCENARIO_CONFIG 即可完成设置，参数范围 0-20，含义见附录说明

- **驱动库接口调用说明**

驱动库接口调用的注意事项如下：

1、 驱动库的接口有些是不能重入的，重入会导致系统异常；
比如硬件读写操作接口：
gh30x_i2c_write_exchange_to_spi()
gh30x_i2c_read_exchange_to_spi()

2、 驱动库的某些接口在退出之前不能被其他接口打断，否则会引起系统异常；
HBD_AdtWearDetectStart()

3、 中断函数不能被打断，否则修改了全部变量，导致整个流程错乱。
gh30x_int_msg_handler()

建议：心率的所有接口调用，推荐采用单任务方式，心率的所有接口调用都在一个任务中执行，中断也建议通过消息发给单线程任务来执行中断函数体；对于多任务的调用，建议对部分接口加互斥锁。

问题反馈

5 版本记录

表 5-1 修订记录

文件版本	修订日期	修订内容
V0.1	2018-09-25	预发布版
V0.2	2018-11-30	预发布版
V0.3	2019-03-13	预发布版
V0.4	2019-07-03	预发布版
V2.0	2020-07-20	根据公版 example 进行重构
V2.1	2020-08-18	修正部分描述
V2.3	2020-09-29	修正部分描述
V3.0	2021-04-29	重构文档，增加驱动 API 说明
V3.1	2021-08-05	由公版裁剪为单驱动版本