

用户指南

——CN131_SDK

负责人 zhousq

版本 2.0.1

更新时间 2025 年 11 月 24 日

修订历史:

2023 年 8 月 4 日——zhangks——1.0.0: Lib 使用说明, 配置移植说明

2023 年 8 月 5 日——zhangks——1.0.1: 更新医标选项说明, 代码架构更新

2023 年 8 月 5 日——zhangks——1.0.2: 更新 mode_define 说明, 代码架构更新

2023 年 8 月 18 日——zhangks——1.0.3: CN131_Parameter_Init 增加一个 VCM 参数, 增加 Lead status enum 数据类型

2023 年 10 月 30 日——zhangks——1.0.4: 修改一些错误

2023 年 11 月 15 日——zhangks——1.0.5: 增加用户问题解答章节

2023 年 11 月 29 日——zhangks——1.0.6: 增加实时心率识别算法 realBeats

2024 年 1 月 11 日——zhangks——1.0.7: 增加对多任务系统的迁移建议

2024 年 1 月 23 日——zhangks——1.0.8: 修改了某些接口

2024 年 1 月 26 日——zhangks——1.0.9: 补充 SPI 说明

2024 年 7 月 3 日——zhangks——1.0.10: 增加应用场景定义

2025 年 11 月 24 日——zhousq——2.0.1: 基于 CN131 重构 SDK

目录

一、 介绍	3
1. 特性	3
二、 SDK 代码解析	3
1. CN131_driver	3
2. User	3
3. Example	4
三、 代码修改与移植	4
1. 移植前准备	4
2. 移植主要修改项	4
3. 配置心电芯片正常工作（裸机系统例程）	10
4. 配置心电芯片正常工作（多任务系统）	15
四、 获取关键参数接口函数	17
1. 数据结构说明	17
2. 接口函数说明	18
五、 系统稳定性设计	18
六、 静态库的使用（基于 cubeIDE 平台）	19
1. 将文件加入工程	19
2. 头文件路径配置	19
3. 静态库路径配置	20
七、 问题解答	21
(1) Q: 使用滤波，陷波功能涉及到采样频率，在哪里可以把采样频率告诉 sdk?	21
(2) Q: 如何调整 CN131 的 "High-pass Pole" "Low-pass Pole" "Channel Gain"	21
(3) Q: Delay_us 函数是必须的吗?	21
(4) Q: ADC 的采样率一定要设置为 250Hz 吗，如果想改成其他数值呢?	21
(5) Q: CN131_Init 一直失败，有哪些原因导致的？如何排除?	22
(6) Q: 配置定时器中断，优先级怎么设定?	22
(7) Q: 使用低功耗模式，有什么需要注意的。	22

一、介绍

本文档主要是 CN131_SDK 固件库使用说明做介绍。CN131_SDK 的示例代码基于 STM32L4 平台。用户可参照示例代码迁移到自己的主控平台上。

文档主要包括: (1) SDK 的代码说明; (2) 如何移植代码及移植过程中的注意事项。

1. 特性

- ADC 采样率默认 250Hz
- ADC 的分辨率 12bit 及以上
- SPI 通信为 GPIO 软件模拟和硬件 SPI 两种可选
- 可拓展支持不同场景的实时心率等计算

二、SDK 代码解析

1. CN131_driver

CN131_driver 文件夹包含 CN131_API.h 和 CN131_API.c。CN131_API.h 提供了一些和 CN131 配置相关的函数, 包括 CN131 初始化, 开启 CN131、控制 CN131 进入不同模式和读取 ECG 信号数据的函数接口。

主要函数和变量如下:

- ① lod_status: 导联脱落状态变量, 值有 LEAD_OFF 和 LEAD_ON 两种。
- ② CN_Heart_Rate: 全局变量, 实时心率 (如需开通相关功能, 请与 CyzurTech 联系)。
- ③ CN131_Parameter_Init: 初始化 CN131_SDK 参数, 包括应用场景, ADC 的分辨率 (默认为 12bit) 和 ADC 的参考电压 (默认为 3.3V)。
- ④ CN131_Init: CN131 初始化函数。
- ⑤ CN131_Start: CN131 启动函数。
- ⑥ CN131_LOD_ON_Init: 导联初始化函数。
- ⑦ CN131_Data_Proc: 数据处理函数。
- ⑧ CN131_Stop: 终止函数, CN131 进入 Stop 模式。
- ⑨ CN131_Standby: CN131 进入 Standby 模式函数。

2. User

CN131_API.c CN131_API.h 文件中, 在注释分隔符 (User Board) 指示下, 包含一些需要

用户自行实现的板级驱动代码，可配置的宏定义和相关 GPIO 初始化、软件 SPI 初始化、ADC 初始化以及定时器中断初始化函数。提供两种方式，一种为用户在自己的工程中使用同名的定义及实现，编译器会自动使用用户定义函数，另一种为在 API.c 文件中对应函数中添加实际功能实现。**用户需要根据自己的主控平台改写。注意某些函数名（注释中带 Wrapper 字段）不可更改，因为驱动中会引用到。**

主要函数和变量如下（这些初始化相关函数名都可修改）：

- ① GPIO_LOD_Init: 芯片 lod 相关 pin 初始化。
- ② GPIO_RST_Init: 芯片 rst 相关 pin 初始化。
- ③ GPIO_SPI_Init: 芯片 spi 相关 pin 初始化。
- ④ SystemClock_Config: 系统时钟配置。
- ⑤ TIM2_Init: TIM 初始化函数。
- ⑥ ADC1_Init: ADC1 初始化函数。

3. Example

Example 文件夹包含了 main.c 例程，该例程为使用 CN131_API.h 中的函数接口，配置 CN131 正确开启工作和采集数据的样例。该例程为裸机系统下的实现，用户可参考修改为多任务系统下的机制。

三、代码修改与移植

1. 移植前准备

首先将 CN131_driver 文件夹下的两个文件 CN131_API.h 和 CN131_API.c 添加到现有工程中，并在编译选项中链接到该静态库。

最后参照 Example 文件夹下的 main.c 例程，配置 CN131 正确开启工作和采集数据。

2. 移植主要修改项

(1) 时钟频率设置

请根据平台需要设置。这里时钟推荐大于 32 MHz。低于该频率 ADC 采样效果不佳。

(2) ADC 采样率设置

CN131_SDK 需要 ADC 的采样率设置为 250Hz。用户可根据自己平台进行相应配置。

以下是 CN131_SDK 基于 ST 平台的配置逻辑，供参考：

CN131_SDK 的 ADC 工作逻辑为软件触发、单次采样。在定时器 TIM 的中断开启 ADC，然后在 ADC 的中断中取值。定时器 TIM 设置为 4ms 产生一次中断，每 4ms 开启一次 ADC，实现 ADC 采样率为 250Hz。ADC 的采样率就是 TIM 的中断频率。

CN131_SDK 的 TIM 的输入时钟频率 Tclk 为系统时钟 80MHz，分频系数 Prescaler 为 159，自动装载值 Period 为 1999。那么 TIM 的溢出时间 Tout 可根据计算公式：

$$Tout = ((Period + 1) * (Prescaler + 1)) / Tclk;$$

其中：

Tclk：TIM3 的输入时钟频率（单位为 Mhz）。

Tout：TIM3 溢出时间（单位为 us）。

计算出 CN131_SDK 的 Tout 为 4000us。

ADC 采样率的配置需要四个步骤：

（1）根据系统时钟计算出 TIM 的分频系数和自动装载值，使得 TIM 的溢出时间为 4000us。并在 TIM 初始化函数 TIM2_Init() 配置好。

（2）在 ADC 初始化函数 ADC_Init() 中将 ADC 的触发方式配置为软件触发，（ExternalTrigConv = ADC_SOFTWARE_START）。

（3）在定时器中断服务函数中开启 ADC，在 ADC 中断服务函数中取值。

```
1. void ADC1_IRQHandler(void)
2. {
3.     HAL_ADC_IRQHandler(&hadc1);
4. }
5. void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
6. {
7.     int16_t adc_val_temp = (int16_t)(HAL_ADC_GetValue(&hadc1) & 0x0000ffff);
8.     CN131_Data_Proc(ADC_IT_PROC, &adc_val_temp);
9. }
10. void TIM2_IRQHandler(void)
11. {
12.     int16_t temp0 = 0;
13.     CN131_Data_Proc(LOD_IT_PROC, &temp0);
14.     HAL_ADC_Start_IT(&hadc1);
15.     HAL_TIM_IRQHandler(&htim2);
16. }
```

（3）应用场景、ADC 分辨率、ADC 参考电压设置、VCM 电压设置

- **应用场景:** CN131_SDK 提供多种场景的参数配置。CN131_Parameter_Init 函数的第一个参数 USE_CASE, 保留参数, 程序中暂不做区分。
- **ADC 分辨率:** CN131_SDK 目前仅支持 12bit ADC 分辨率, 若需支持更改分辨率 ADC 采样请联系 CyzurTech。
- **ADC 参考电压:** CN131_SDK 支持更改 ADC 的参考电压。CN131_Parameter_Init 函数的第三个参数为 ADC 参考电压, 当前 SDK 中 VREF 的值为 3.3V。
- **VCM 电压:** CN131_SDK 支持更改 VCM 的电压值。CN131_Parameter_Init 函数的第四个参数为 VCM 电压, 是芯片 VCMpin 的供电电压, 当前 SDK 中 VCM 的值为 0.9V。

```
1.  /**
2.   * @brief
3.   * @note  USE_CASE: application scenarios
4.   * @note  ADC_RESOLUTION
5.   * @note  ADC12BIT: Default. ADC RESOLUTION == 12bit.
6.   * @note  VREF ADC Reference Voltage. Default 3.3
7.   * @note  VCM Voltage. Default 0.9
8.   */
9.  void CN131_Parameter_Init(USE_CASE use_case, ADC_RESOLUTION adc_resolution,
    float VREF, float VVCM);
```

若将 CN131 应用 ADC 分辨率为 12bit, ADC 的参考电压为 3.3V, VCM 电压为 0.9V。应分为两个步骤设置:

(1) 在主函数中, 使用 CN131_Parameter_Init 参数如下。

```
1.  CN131_Parameter_Init(0, ADC12BIT, 3.3, 0.9);
```

(2) 用户根据自身板卡配置 ADC, 保持参数一致即可。

(4) SPI 设置

CN131 芯片的配置通信方式为 SPI, CN131_SDK 驱动中集成了 GPIO 软件模拟 SPI 通信, 用户也可以选择硬件 SPI 外设, 更稳定。

- **GPIO 软件模拟 SPI**

若使用驱动中的 **GPIO 软件模拟 SPI**, 需要对 SPI 信号线组 (4 条: CSB 输入、SCLK 输入、

MOSI 输入、MISO 输出) 进行配置。配置流程如下:

(1) 在宏定义中修改对应的引脚。

```
1. /* Define Daughter Board(DB) SPI pins */
2. #define DB_CSB_Pin GPIO_PIN_4
3. #define DB_CSB_GPIO_Port GPIOA
4. #define DB_SCLK_Pin GPIO_PIN_5
5. #define DB_SCLK_GPIO_Port GPIOA
6. #define DB_MOSI_Pin GPIO_PIN_7
7. #define DB_MOSI_GPIO_Port GPIOA
8. #define DB_MISO_Pin GPIO_PIN_6
9. #define DB_MISO_GPIO_Port GPIOA
```

(2) 修改 GPIO_WritePin 和 GPIO_ReadPin 函数 Wrapper 定义, CN131_SDK 提供的例程为 HAL 库的函数写法, 需要根据平台的不同修改对应的 GPIO_WritePin 和 GPIO_ReadPin 函数。

SPI 的 CSB、SCLK、MOSI 需要配置 WritePin(SET) 和 WritePin(RESET) 两组函数 Wrapper。

SPI 的 MISO 需要配置 ReadPin()==SET 和 ReadPin()==RESET 两组函数 Wrapper。

```
1. /* Wrapper: WritePin Function */
2. void DB_CSB_SET(void);
3. void DB_CSB_RESET(void);
4. void DB_SCLK_SET(void);
5. void DB_SCLK_RESET(void);
6. void DB_MOSI_SET(void);
7. void DB_MOSI_RESET(void);
8. /* Wrapper: ReadPin Function */
9. uint8_t DB_MISO_HIGH(void);
10. uint8_t DB_MISO_LOW(void);
```

(3) GPIO_SPI_Init() 函数中对 GPIO 引脚初始化。

(4) 确保 CN1xx_SPI_ReadWrite 的返回值为 0;

```
1. uint8_t CN1xx_SPI_ReadWrite(uint8_t* TxDataSeq, uint8_t* RxDataSeq, uint8_t SeqLen)
```

```

2.  {
3.    // HAL_SPI_TransmitReceive(&hspi1, TxDataSeq, RxDataSeq, SeqLen, HAL_MAX_DELAY);
4.    uint8_t hw_spi = 0; // if use hardware spi, make sure hw_spi = 1;
5.    return hw_spi;
6.  }

```

● 硬件 SPI 外设

若用户选择硬件 SPI 外设，也可以通过下面四步配置：

- (1) 在宏定义删除 SPI 对应的引脚。避免 pin 脚变为 GPIO 模式。且请保证 hw_spi = 1。
- (2) 修改 GPIO_WritePin 和 GPIO_ReadPin 函数 Wrapper 定义，SPI 的 CSB、SCLK、MOSI 的 WritePin (SET) 和 WritePin (RESET) 两组函数 Wrapper，内部定义留空即可。SPI 的 MISO 的 ReadPin()==SET 和 ReadPin()==RESET 两组函数 Wrapper，另返回值 0 即可。
- (3) 初始化硬件 SPI 外设（全双工）。
- (4) 将硬件 SPI 的 TransmitReceive 函数定义在 CN1xx_SPI_ReadWrite 内部，并确保返回值为 1。CN1xx_SPI_ReadWrite 的三个传参分别代表：TxDataSeq 发送序列数组指针，RxDataSeq 接受序列数组指针，SeqLen 序列共同长度。TxDataSeq 和 RxDataSeq 长度一致。

```

1.  uint8_t CN1xx_SPI_ReadWrite(uint8_t* TxDataSeq, uint8_t* RxDataSeq, uint8_t SeqLen)
2.  {
3.    HAL_SPI_TransmitReceive(&hspi1, TxDataSeq, RxDataSeq, SeqLen, HAL_MAX_DELAY);
4.    uint8_t hw_spi = 1; // if use hardware spi, make sure hw_spi = 1;
5.    return hw_spi;
6.  }

```

(5) GPIO 设置

CN131_SDK 还需要用到两个引脚，分别是 CN131 芯片的 LOD 输出，CN131 芯片的 RST 输入。

用户可根据硬件电路，灵活选择配置。如 RST 悬空，就不配置 RST；应用电路无导联脱落机制，就不配置 LOD。

配置引脚需要三个步骤：

- (1) 在宏定义中修改对应的引脚，不需要的宏定义引脚可以注释掉。

```

1.  /* Define Daughter Board(DB) LOD pin */
2.  #define DB_LOD1_Pin GPIO_PIN_2
3.  #define DB_LOD1_GPIO_Port GPIOA
4.  /* Define Daughter Board(DB) RST pin */

```



```
5. #define DB_RST_Pin GPIO_PIN_5
6. #define DB_RST_GPIO_Port GPIOC
```

(2) 修改 GPIO_WritePin 和 GPIO_ReadPin 函数 Wrapper 定义, CN131_SDK 为 HAL 库的函数写法, 需要根据平台的不同修改对应的 GPIO_WritePin 和 GPIO_ReadPin 函数。

CN131 的 RST 需要配置 WritePin (SET) 和 WritePin (RESET) 两组函数 Wrapper。不需要配置的引脚, 保留函数名, 但内部不定义即可。

CN131 的 LOD1 需要配置 ReadPin()==SET 和 ReadPin()==RESET 两组函数 Wrapper。不需要配置的引脚, 保留函数名, 令返回值为 0 即可。

```
1. /* Wrapper: WritePin Function */
2. void DB_RST_SET(void);
3. void DB_RST_RESET(void);
4. /* Wrapper: ReadPin Function */
5. uint8_t DB_LOD1_LOW(void);
6. uint8_t DB_LOD1_HIGH(void);
```

(3) 在 GPIO_LOD_Init()、GPIO_RST_Init() 函数中对 GPIO 引脚初始化。

```
1. void GPIO_LOD_Init(void)
2. {
3.     GPIO_InitTypeDef GPIO_InitStructure;
4.     __HAL_RCC_GPIOA_CLK_ENABLE();
5.     GPIO_InitStructure.Pin = DB_LOD1_Pin;
6.     GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
7.     GPIO_InitStructure.Pull = GPIO_NOPULL ;
8.     HAL_GPIO_Init(DB_LOD1_GPIO_Port, &GPIO_InitStructure);
9. }
10. void GPIO_RST_Init(void)
```

(6) Delay_us

CN131_API.h 内部函数需要调用一个微秒级的延时, 用于模拟 SPI 的时钟生成。目前 SDK 是使用 HAL 库生成的, 如果移植, 请务必改写成适用平台的写法。若使用软件 SPI 或者 RST

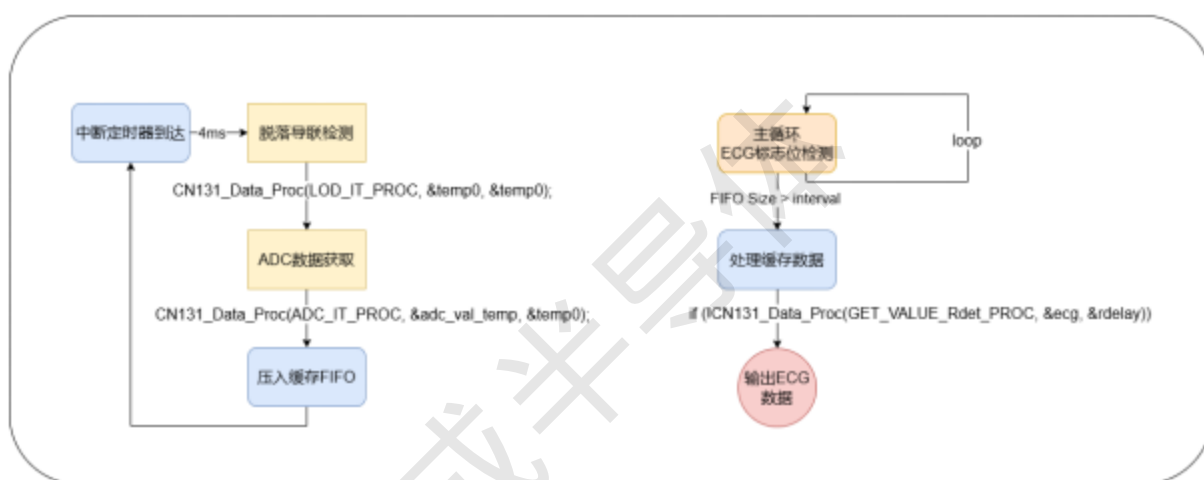
引脚, 该函数内部定义需要改写, 若两者都不需要则内部定义为空即可。

```
1. void CN1xx_delay_us(uint32_t udelay) {...}
```

3. 配置心电芯片正常工作（裸机系统例程）

至此我们已经完成了所有基本功能的配置, 除此之外, 用户还应当配置好串口输出, 来输出心电数据。接下来让我们来逐步调试, 使得芯片可正常工作并输出心电数据。以下例子为评估套件使用的逻辑, 用户可参考配置。

CN131_SDK 输出的 ECG 为滤波后的信号。利用 CN131_API 提供的接口, 可以读取 ecg 信号。例程为裸机系统下的轮询机制, 主要逻辑如下图所示。



以 CN131_SDK 的 main.c 说明配置方法, 请结合 main.c 阅读。步骤如下:

(1) 配置 ADC 中断服务函数

配置 `ADC1_IRQHandler` 和 `HAL_ADC_ConvCpltCallback` 函数, 当前 SDK 为 HAL 库, 需要修改为移植平台支持的中断服务函数。

`CN131_Data_Proc()` 函数: `ADC_IT_PROC` 工作模式时用在 ADC 中断的 callback 函数, 将 ADC 的值通过第二个参数传入。其返回值有两种情况:

- 0: 代表数据存储成功;
- 1: 代表内部 BUFFER 已满, 存储失败。

我们要重点关注 result=1 的情况, 这代表 mcu 处理 ecg 数据（取数滤波等操作）的速度跟不上 ADC 采集的速度。此时会丢失掉部分数据, 需要精简处理数据部分的代码, 或者提高主频确保处理速度和采集速度匹配。

```
1. void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
```

```
2. {  
3.     int16_t adc_val_temp = (int16_t)(HAL_ADC_GetValue(&hadc1) & 0x000fffff);  
4.     CN131_Data_Proc(ADC_IT_PROC, &adc_val_temp);  
5. }
```

此过程是将 ADC 的数据压入驱动内部的 FIFO。某些主控平台的 ADC 没有中断机制，可在定时器中断服务函数使用此函数，只要保证 4ms（250hz 采样率，用户根据实际需要采样率）将一个 ADC 采集的数据压入内部缓存即可。

（2）配置定时器中断服务函数

在此中断服务函数中需要调用两个函数：

1) **ADC_START_IT()**，用 TIM 的中断软件触发 ADC

2) **CN131_Data_Proc()**函数：使用 **LOD_IT_PROC** 工作模式，在这个过程中，由第二个和第三个参数在该工作模式下并未使用。

其返回值在该模式下仅返回 0。

```
1. void TIM2_IRQHandler(void)  
2. {  
3.     int16_t temp0 = 0;  
4.     CN131_Data_Proc(LOD_IT_PROC, &temp0, &temp0);  
5.     HAL_ADC_Start_IT(&hadc1);  
6.     HAL_TIM_IRQHandler(&htim2);  
7. }
```

（3）基本外设初始化

在主函数最开始部分，进行 **ADC、TIM 和 GPIO** 等初始化。建议将 **CN131** 相关内容封装成一个函数进行初始化，在特殊情景下如低功耗，或 **CN131** 波形不正常等情况下可重新初始化。

```
1. SystemClock_Config();  
2. ADC1_Init();  
3. TIM2_Init();  
4. GPIO_LOD_Init();
```

```
5. GPIO_RST_Init();  
6. GPIO_SPI_Init();  
7. USART1_UART_Init();
```

(4) 配置 CN131 基本参数

然后使用 `CN131_Parameter_Init` 设置 CN131 的参数, 当前参数代表 CN131 应用在手持式设备, ADC 分辨率为 12bit, ADC 的参考电压为 3.3V, 芯片的参考电压 `Vref` 为 0.9V。

```
1. CN131_Parameter_Init(0, ADC12BIT, 3.3, 0.9);
```

(5) 配置 CN131 工作模式

CN131_SDK 提供 `Medical_MODE` 模式和 `Health_MODE` 模式, 通过使用 `CN131_API.h` 中的函数 `CN131_Init(CN131_WORKMODE work_mode)` 配置, 可为 CN131 选择不同的工作模式: `Medical_MODE` or `Health_MODE`。

函数返回 `uint8_t` 类型值表明 Initial 结果, 1 代表成功, 0 代表失败。**务必要确保该函数可配置成功, 若配置失败, 芯片也可正常输出心电波形 (此时是使用芯片的上电默认配置), 但波形质量可能不高。**

此函数调用的 SPI 对芯片配置。若返回值失败, 请检查硬件 SPI 的写法。如果使用的是软件 SPI 模拟, 原因可能有三:

- 1、GPIO 配置的模式不正确;
- 2、GPIO 读写高低电平配置不正确;
- 3、`cn1xx_delay_us` 延时写法不正确。可以根据这三个原因逐步确认。

```
1. If(CN131_Init(Medical_MODE)) { /*printf("success!\n");*/ }
```

(6) 导联脱落功能设置

`CN131_Data_Proc()` 函数: 使用 `LOD_IT_PROC` 工作模式, 就会每个定时器中断中检测 CN131 芯片导联是否脱落, 变量 `lod_status` (变量值在驱动中会自动更新) 表明导联的状态。

注: 在脱落导联函数判断时, 由于工频干扰需要计次判断, 这里提供 2 个宏定义, 用户可以根据实际情况调整:

```

1. // LEAD ON
2. #define LEADON_JUDGE_LIMIT 3 // UNIT: ADC_UPDATE_FREQ
3. #define LEADOFF_JUDGE_LIMIT 4 // UNIT: ADC_UPDATE_FREQ

```

lod_status 值为 LEAD_OFF 代表脱落, lod_status 值为 LEAD_ON 代表导联。当 lod_status 离开脱落状态后, 使用 **CN131_LOD_ON_Init()** 函数进行导联状态初始化。

CN131_LOD_ON_Init() 函数, 建议将 ECG 相关内容封装为一个函数, 调用示例如下:

```

1. void ecg_function_init()
2. {
3.     GPIO_LOD_Init();
4.     GPIO_RST_Init();
5.     GPIO_SPI_Init();
6.     GPIO_LDO_Init();
7.     DB_LDO_SET();
8.     delay_ms(500);
9.     CN131_Parameter_Init(0, ADC12BIT, 3.6, 0.9);
10.    if (CN131_Init(Health_MODE)){NRF_LOG_INFO("\r\n CN131_Init success \r\n");}
11.    else{ NRF_LOG_INFO("\r\n CN131_Init failed \r\n"); }
12.    CN131_LOD_ON_Init();
13.}

```

注: 1、若用户芯片需要配置低功耗, 可定时获取导联状态。进行低功耗条件, 示例代码如下:

```

1. void timer_ecg_send_timeout_handler(void *p_context)
2. {
3.     static uint32_t leadoffcount = 0;
4.     static uint8_t lod_pre_status = LEAD_OFF;
5.     if (lod_status == LEAD_ON)
6.     {
7.         if (lod_pre_status == LEAD_OFF){ CN131_LOD_ON_Init(); }

```

```
8.         lod_pre_status = LEAD_ON; leadoffcount = 0; flag_lead_off_timeout = 0;
9.     }
10.    else if (lod_status == LEAD_OFF)
11.    {
12.        lod_pre_status = LEAD_OFF;
13.        leadoffcount++;
14.        if (leadoffcount >= LEAD_OFF_TIMEOUT){ flag_lead_off_timeout = 1; }
15.        if (leadoffcount >= SLEEP_TIMEOUT)
16.        { leadoffcount = 0; sleep_mode_enter(); }
17.    }
18. }
```

(7) 配置读取心电数据

`CN131_Data_Proc()`函数共有三种工作模式，分别是：`LOD_IT_PROC`、`ADC_IT_PROC`和`GET_VALUE_Rdet_PROC`。前文已经用到了前两种模式。接下来重点介绍，后一种模式。

使用`GET_VALUE_Rdet_PROC`模式采集心电信号 `ecg` 的数据和实时心率（如需开通相关功能，请与 CyzurTech 联系）。

使用时，需要新建一个变量 `ecg`，以指针的形式传入取心电信号的值。新建一个变量 `Rdelay`，来获取此时识别到的 R 峰与当前点的 `delay`（识别 R 峰有延时）。

当 `Rdelay` 为 0 时，表明没有识别到 R 点；当 `Rdelay` 不为 0 时，表明识别到 R 点，同时全局变量 `heart_Rate` 会更新，基于当前 RR 间期计算实时心率。

采集处理数据的过程中，驱动内部使用了一个缓存，`CN131_Data_Proc()`函数工作在`GET_VALUE_Rdet_PROC`模式下时，返回值共有三种情况：

0: `get_value` 成功；

1: `get_value` 失败，原因是缓存已空，这代表 `mcu` 处理 `ecg` 数据的速度超过了 `ADC` 采集的速度；

2: `get_value` 失败，原因是缓存已满，这代表 `mcu` 处理 `ecg` 数据的速度跟不上 `ADC` 采集的速度。

我们要重点关注 `result=2` 的情况，此时会丢失掉部分数据，需要精简处理数据的程序。

或者提升主频减少主循环运行时间。

```
1. while(led_status == LEAD_ON) {
2.     int16_t ecg, Rdelay;
3.     uint8_t result = CN131_Data_Proc(GET_VALUE_PROC, &ecg, &Rdelay);
4.     if(result == 0) {
5.         printf("ecg value = %d, heart_Rate = %d\r\n", ecg, heart_Rate);
6.     }
7. }
```

(8) 此外还有部分函数接口在 **main** 函数实例中并没有给出，在这里做出说明：

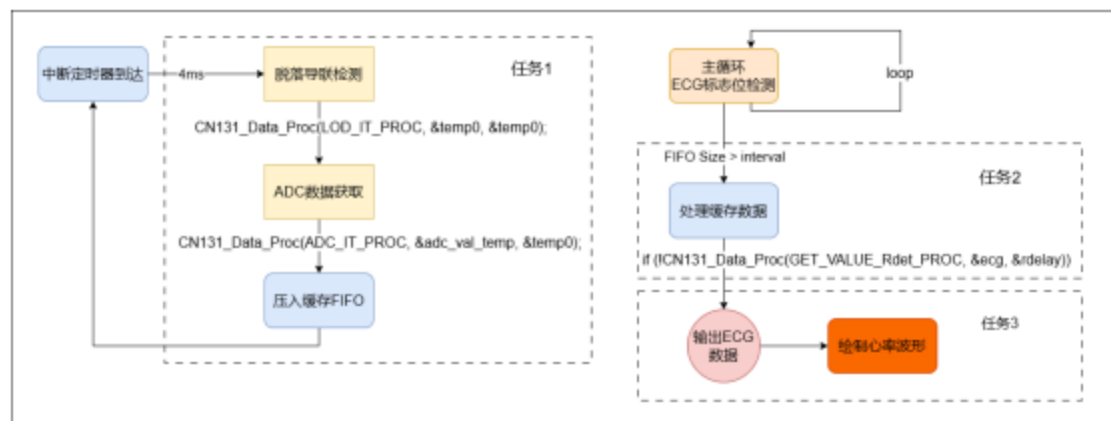
CN131_Standby()函数可以使芯片进入 **Standby** 模式（具体含义见 **datasheet**）。

CN131_Stop()函数使芯片进入 **power off** 模式。

4. 配置心电芯片正常工作（多任务系统）

至此我们已经熟悉了所有基本功能的配置，除此之外，用户的实际平台大多是在实时操作系统下开发，这里给出主要函数功能说明和逻辑配合，用户可根据实际的业务需要，进行适当调整。用户可先阅读裸机系统的例程，理解主要函数的运行逻辑，然后再移植。

下图是 **CN131_SDK** 的运行逻辑，ADC 采样的 ECG 数据送入缓存，然后从缓存中取出 ECG 数据滤波并计算实时心率，最后使用滤波后的 ECG 数据进行绘图或蓝牙发送到上位机。除此之外还有一个任务进行实时脱落检测和快速恢复。



我们需要关注 CN131_Data_Proc 函数在不同阶段下的操作。

- **中断 (4ms)**: 这部分函数是将 ADC 采集的数据每隔 4ms 送进内部缓存, 以保证 250Hz 采样。

- **注: 用户在配置中断时, 需要保证该定时器中断优先级最高, 否则会导致数据采样异常。**

若用户平台有其他方法可以保证采样率为 250Hz, 也可以, 保证把数据传入该函

CN131_Data_Proc(ADC_IT_PROC,...) 就行。

- **任务 1:** 这个任务中的函数 CN131_Data_Proc(LOD_IT_PROC, ...) 是定时去检测导联脱落状态、同时执行芯片的快速恢复功能。同时对 ADC 进行采集, 采集的数据压入 FIFO 中。

该任务需要是个定时任务 (或中断), 大概每隔 4ms (或稍大些间隔时间) 的时间执行一次。

- **任务 2:** 这个任务 (或中断) 中的函数 CN131_Data_Proc(GET_VALUE_Rdet_PROC, ...) 是执行信号滤波、计算心率等功能。用户可通过 “FIFO_SIZE()” 接口获取缓存内的数据量。

用户可自行选择缓存内的数据量到达某一个阈值后, 然后执行该任务。示例代码:

```
1. int16_t adc_val_temp = (int16_t)saadc_val;
2. CN131_Data_Proc(ADC_IT_PROC, &adc_val_temp, &temp0);
3. if (FIFO_Size() >= ECG_PROCESS_INTERVAL)
4. {
5.     ecg_full = 1;
6. }
```

- **任务 3:** 这个任务是用户自定义的绘图或蓝牙发送数据等函数功能, 如果有定制化需求, 也可以联系 CyzurTech。为了不影响处理数据的及时性, 该任务的优先级应该要比任务 2 要低一些。

该任务与任务 2 的接口也可以通过一个环形数组, 进行数据的暂存, 保证数据不丢失。

如运算速度比较快的情况下, 该任务也可与任务 2 合并, 处理好数据后立即发送或绘图。

注意:

- (1) 任务 1 中执行的函数调用了 SPI 接口, 如果使用 SDK 内部的软件模拟 SPI, 若该任

务被其他任务打断, 则可能 SPI 信号不完整。如果该函数返回值异常, 可以使用硬件 SPI 解决。

(2) 任务 1 中执行了两个功能, 导联脱落检测和快速恢复, 也可以把这两个功能分别在两个任务中执行, 可以通过控制 CN131_Data_Proc(LOD_IT_PROC, ...) 中的不同参数来配置。

四、获取关键参数接口函数

1. 数据结构说明

```
1. typedef struct CN1xx_API_Para_t
2. {
3.     uint16_t magic;
4.     uint8_t res[4];
5.     uint8_t work_mode;
6.     uint8_t lead_status;
7.     uint8_t gain;
8.     uint8_t hp;
9.     uint8_t lp;
10.    uint8_t auto_fr_status;
11.    ErrorCode error_code;
12.    uint8_t version[2];
13.    float adc_vol_det;
14.} CN1xx_API_Para_t;
15.
```

这里使用了结构体对 CN131 相关关键参数进行了封装, 并提供了接口函数进行传递。确保内存中内部变量稳定。

2. 接口函数说明

```
1. CN1xx_APIuint32_t Get_CN1xx_parameter(CN1xx_API_Para_t *parameter)
2. {
3.     memcpy(parameter, &g_para, sizeof(CN1xx_API_Para_t));
4.     return sizeof(CN1xx_API_Para_t);
```

5. }

该接口函数有返回值，返回值为结构体大小，需要传入指针，**调用该接口后，用户可以再封装一层，由串口或蓝牙等接口发送到上位机，这样就能解析出运行参数。**

五、系统稳定性设计

基于工程稳定性的设计，这里有点已知系统性问题，在这里着重说明：

1、由于采样需要高实时性，在定时器中断中，我们有 SPI 任务和 ADC 任务，这两个任务都需要不被打断。所以**建议将该定时器中断抢占优先级和响应优先级都设置为最高。**

2、若客户有低功耗需求，基于 sleep、stop、standby 这三种 MCU 低功耗模式，这里说明程序设计框架：

(1) standby 模式，由于唤醒后重新执行的入口是 main 函数，则会重新初始化相关外设和函数。则不需额外做逻辑判断。基于这三种低功耗模式，**最推荐 standby 模式。**

(2) stop 模式，该模式下进入低功耗模式后，会关闭时钟，则恢复后需要重新配置时钟，以及重新初始化外设。则需要重新初始化 CN131 相关函数。

(3) sleep 模式，该模式下虽不关闭时钟，但休眠了 CPU，且定时器到达后相关任务需 CPU 的参与运行。需要根据睡眠条件以及唤醒条件重新设计采集/处理任务等。

六、静态库的使用（基于 cubelIDE 平台）

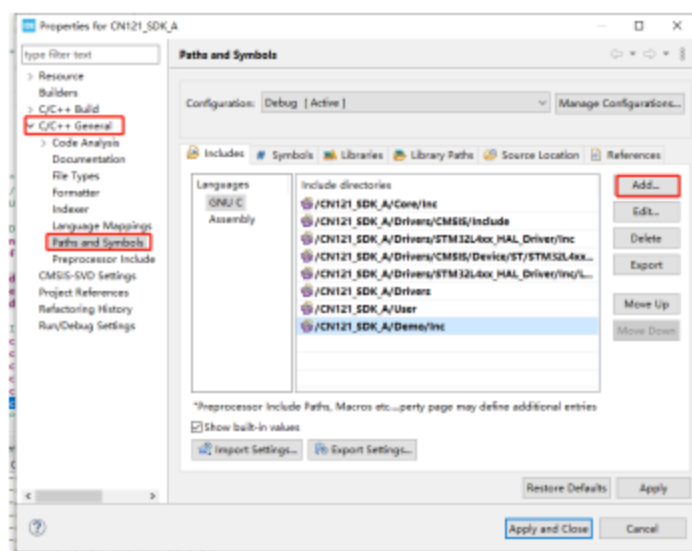
1. 将文件加入工程

直接将 CN131 文件夹复制进 Cube 工程目录下即可，在工程中就会自动将其加入工程。

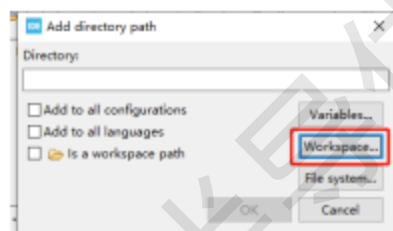
2. 头文件路径配置

点击菜单栏“Project”下的“Properties”选项，打开工程的选项设置，按照如下步骤添加头文件目录。

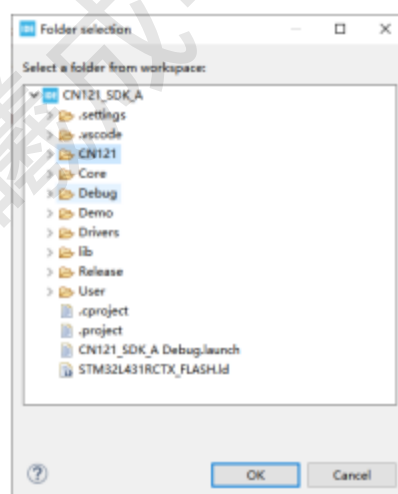
选中“C/C++ General”下的“Paths and Symbols”，点击“Add”添加路径。



点击“Workspace”。

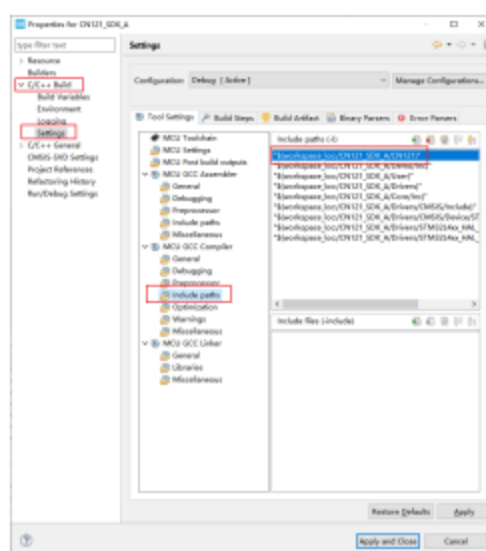


选中 CN131 文件夹并确认，点击“apply”，路径添加成功。

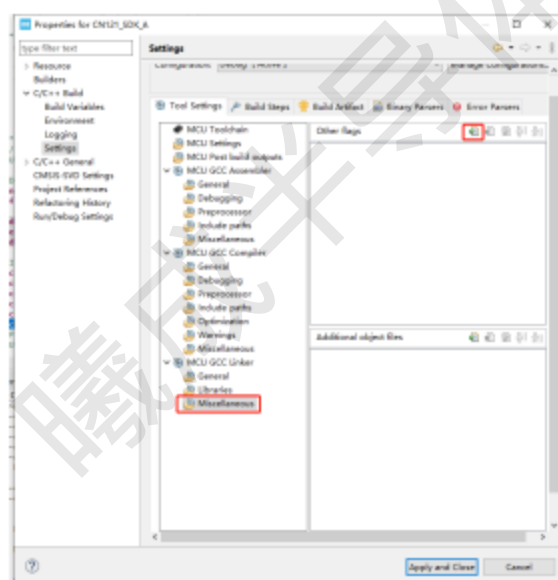


3. 静态库路径配置

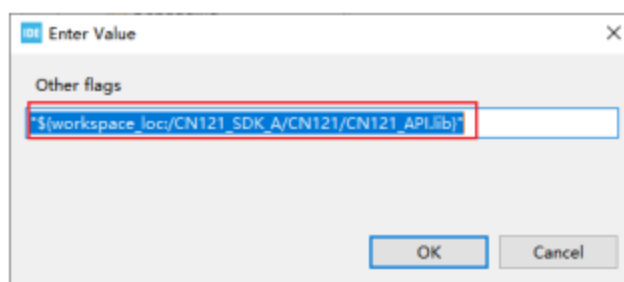
按图中步骤点击，选中 CN131 的 Inc 文件路径，双击打开后复制这个路径。



点击“MCU GCC Linker”下的“Miscellaneous”，添加 lib 文件的路径。



将刚刚复制的路径粘贴进去，并在“CN131”后面添加“/CN131_API.lib”。



最后点击确认并应用。至此，lib 文件添加成功。

最后保存配置，重新编译就 OK 了。

七、问题解答

(1) **Q: 使用滤波, 陷波功能涉及到采样频率, 在哪里可以把采样频率告诉 sdk?**

A: 目前的驱动内部的滤波默认是 250HZ 采样率, 无需这一步操作。

(2) **Q: 如何调整 CN131 的 "High-pass Pole" "Low-pass Pole" "Channel Gain"**

A: 驱动中有封装好的两套配置 (实验下来心电波形效果比较好), medical 和 Health 模式。用户只需要调用函数 CN131_Init (WORK_MODE) 配置使用即可。不用再另外配置 AFE 的带宽增益设置。

(3) **Q: Delay_us 函数是必须的吗?**

A: 驱动中是使用 GPIO 软件模拟 SPI, SPI 的时钟信号需要用到此函数生成。SPI 主要是对 CN131 进行参数配置, 若对此过程无很大的时间要求。可以用毫秒级延时, 但不要改变 Delay_us 的函数名, 可在函数内部再嵌套一层。

(4) **Q: ADC 的采样率一定要设置为 250Hz 吗, 如果想改成其他数值呢?**

A: 驱动内部的滤波参数默认是 250HZ 采样率, 如果想改成其他数值可能会影响滤波效果, 影响心电波形质量。可联系 CyzurTech 技术支持重新定制驱动。

(5) **Q: CN131_Init 一直失败, 有哪些原因导致的? 如何排除?**

A: 这一步是使用 SPI 对 CN131 进行配置。在驱动中, 是使用 GPIO 软件模拟的 SPI 通信。所以第一步检查 GPIO 的模式有没有配置正确, 可参考 GPIO_SPI_Init() 中的配置或者用户平台 SPI 功能对于 GPIO 推荐的配置, 对于输出, 能正确写高低电平。对于输入, 能正确读取高低电平。第二步使用逻辑分析仪, 看是否有对应波形出来。第三步, 软件模拟时钟 SCLK 的信号, 使用到 Delay_us 函数, 此函数是否设置正确。

(6) **Q: 配置定时器中断, 优先级怎么设定?**

A: 由于 ADC 采集数据与算法有强实时性, 需要保证采样频次为稳定的间隔, 且不要被其他中断所打断。则应设定为中断向量表中最高优先级。

(7) **Q: 使用低功耗模式, 有什么需要注意的。**

A: 若使用的是 standby 模式, 由于唤醒后重新执行的入口是 main 函数, 则会重新初始化相关外设和函数。最推荐。则不需额外做逻辑判断。

若使用 stop 模式, 唤醒后, 需重新初始化 CN131, 且重新配置时钟, 不推荐。

若使用 sleep 模式进行低功耗, 唤醒后也需重新初始化 CN131 设备。