用户指南

——CN121_SDK

负责人 zhangks

版本 C.4

更新时间 2024 年 7 月 3 日

修订历史：

2023 年 8 月 4 日——zhangks——B.1.0: Lib 使用说明，配置移植说明

2023 年 8 月 5 日——zhangks——B.2.0: 更新医标选项说明，代码架构更新

2023 年 8 月 5 日——zhangks——B.3.0: 更新 mode_define 说明，代码架构更新

2023 年 8 月 18 日——zhangks——B.4.0: CN121_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——B.5.0: 修改一些错误

2023 年 11 月 15 日——zhangks——B.6.0: 增加用户问题解答章节

2023 年 11 月 29 日——zhangks——C.1.0: 增加实时心率识别算法 realRhythm

2024 年 1 月 11 日——zhangks——C.2.0: 增加对多任务系统的迁移建议

2024 年 1 月 23 日——zhangks——C.2.1: 修改了某些接口

2024 年 1 月 26 日——zhangks——C.2.2: 补充 SPI 说明

2024 年 7 月 3 日——zhangks——C.4: 增加应用场景定义

目录

一、介绍

本文档主要是 CN121_SDK 固件库使用说明做介绍。 CN121_SDK 的示例代码基于 STM32L4 平台，用户拿到手的 SDK 中的核心静态库已针对用户的主控平台做了适配。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 采样率为 250Hz
- ADC 的分辨率适配 12bit 和 10bit
- 工频滤波为 50Hz
- 支持 Health 模式和 Medical 模式
- ECG 信号波形有医标和非医标两种可配置选项
- SPI 通信为 GPIO 软件模拟和硬件 SPI 两种可选
- 支持实时识别 R 峰，输出心率

二、SDK 代码解析

1. CN121_driver

CN121_driver 文件夹包含 CN121_API.h 和 CN121_API.lib。CN121_API.h 提供了一些和 CN121 配置相关的函数，包括 CN121 初始化，开启 CN121、控制 CN121 进入不同模式和读取 ECG 信号数据的函数接口。CN121_API.lib 为基于用户主控平台内核打包生成的静态库。

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。

③ CN121_Parameter_Init：初始化 CN121_SDK 参数，包括应用场景，ADC 的分辨率选项（12bit 或 10bit）和 ADC 的参考电压（默认为 3.3V）。

④ CN121_Init：CN121 初始化函数。

⑤ CN121_Start：CN121 启动函数。

⑥ CN121_LOD_ON_Init：导联初始化函数。

⑦ CN121_Data_Proc：数据处理函数。

⑧ CN121_Stop：终止函数，CN121 进入 Stop 模式。

⑨ CN121_Standby：CN121 进入 Standby 模式函数。

2. User

User 文件夹下为 CN121_utils.h 和 CN121_utils.c。包含一些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。**该部分适配 STM32L4 平台的 HAI 库，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 CN121_API.h 中的函数接口，配置 CN121 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 CN121_driver 文件夹下的两个文件 CN121_API.h 和 CN121_API.lib 添加到现有工

程中，并在编译选项中链接到该静态库。

然后将 User 文件夹下的 CN121_utils.h 和 CN121_utils.c，添加到现有工程中。移植过程中主要对这两个文件进行修改。

最后参照 Example 文件夹下的 main.c 例程，配置 CN121 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。

（2）ADC 采样率设置

CN121_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 CN121_SDK 基于 ST 平台的配置逻辑，供参考：

CN121_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。

CN121_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 CN121_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.      CN121_Data_Proc(ADC_IT_PROC, &adc_val_temp);
9.  }
10. void TIM2_IRQHandler(void)
11. {
12.     int16_t temp0 = 0;
13.     CN121_Data_Proc(TIMER_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：**CN121_SDK 提供多种场景的参数配置。CN121_Parameter_Init 函数的第一个参数 USE_CASE 有三个选项：

  USE_FOR_STANDARD：满足医标；

  BODY_TOUCH：用于与身体接触场景，如心电贴；

  HANDHELD：用于手持式场景，如单导心电卡。

- **ADC 分辨率：**CN121_SDK 支持两档 ADC 分辨率，分别为 10bit 和 12bit。CN121_Parameter_Init 函数的第二个参数 ADC_RESOLUTION，选择“ADC12BIT”为 12bit，“ADC10BIT”为 10bit。

- **ADC 参考电压：**CN121_SDK 支持更改 ADC 的参考电压。CN121_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：**CN121_SDK 支持更改 VCM 的电压值。CN121_Parameter_Init 函数的第四个参数为 VCM 电压，是芯片 VCM pin 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       USE_FOR_STANDARD: Meet to clinical criteria;
5.  * @note       BODY_TOUCH: Apply to the scenario of touch with human body;
6.  * @note       HANDHELD: Apply to the handheld product.
7.  * @note   ADC_RESOLUTION
8.  * @note      ADC10BIT: ADC RESOLUTION == 10bit;
9.  * @note      ADC12BIT: Default. ADC RESOLUTION == 12bit.
10. * @note   VREF ADC Reference Voltage. Default 3.3
11. * @note   VCM Voltage. Default 0.9
12. */
13. void CN121_Parameter_Init(USE_CASE use_case, ADC_RESOLUTION adc_resolution,
    float VREF, float VVCM);
```

若将 CN121 应用为手持式场景，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 CN121_Parameter_Init 参数如下。

```
1.  CN121_Parameter_Init(HANDHELD, ADC12BIT, 3.3, 0.9);
```

（2）在 CN121_utils.h 和 CN121_utils.c 中， ADC1 初始化函数 ADC1_Init()中将 ADC 的“Resolution”配置为 12bit。

（4）SPI 设置

CN121 芯片的配置通信方式为 SPI，CN121_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。

```
/* Define Daughter Board(DB) SPI pins */
#define DB_CSB_Pin GPIO_PIN_4
#define DB_CSB_GPIO_Port GPIOA
#define DB_SCLK_Pin GPIO_PIN_5
#define DB_SCLK_GPIO_Port GPIOA
#define DB_MOSI_Pin GPIO_PIN_7
#define DB_MOSI_GPIO_Port GPIOA
#define DB_MISO_Pin GPIO_PIN_6
#define DB_MISO_GPIO_Port GPIOA
```

（2）修改 GPIO_WritePin 和 GPIO_ReadPin 函数 Wrapper 定义， CN121_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。

```
/* Wrapper: WritePin Function */
void DB_CSB_SET(void);
void DB_CSB_RESET(void);
void DB_SCLK_SET(void);
void DB_SCLK_RESET(void);
void DB_MOSI_SET(void);
void DB_MOSI_RESET(void);
/* Wrapper: ReadPin Function */
uint8_t DB_MISO_HIGH(void);
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；

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

```
{
//  HAL_SPI_TransmitReceive(&hspi1, TxDataSeq, RxDataSeq, SeqLen, HAL_MAX_DELAY);
    uint8_t hw_spi = 0; // if use hardware spi, make sure hw_spi = 1;
    return hw_spi;
}
```

- **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。

（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，另返回值为 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。

```
uint8_t CN1xx_SPI_ReadWrite(uint8_t* TxDataSeq,uint8_t* RxDataSeq,uint8_t SeqLen)
{
    HAL_SPI_TransmitReceive(&hspi1, TxDataSeq, RxDataSeq, SeqLen, HAL_MAX_DELAY);
    uint8_t hw_spi = 1; // if use hardware spi, make sure hw_spi = 1;
    return hw_spi;
}
```


（5）GPIO 设置

CN121_SDK 还需要用到两个引脚，分别是 CN121 芯片的 LOD 输出，CN121 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。

```
/* Define Daughter Board(DB) LOD pin */
#define DB_LOD1_Pin GPIO_PIN_2
#define DB_LOD1_GPIO_Port GPIOA
```

```
/* Define Daughter Board(DB) RST pin */
#define DB_RST_Pin GPIO_PIN_5
#define DB_RST_GPIO_Port GPIOC
```

（2）修改 GPIO_WritePin 和 GPIO_ReadPin 函数 Wrapper 定义， CN121_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**。

SPI 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。

SPI 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。

```
/* Wrapper: WritePin Function */
void DB_RST_SET(void);
void DB_RST_RESET(void);
/* Wrapper: ReadPin Function */
uint8_t DB_LOD1_LOW(void);
uint8_t DB_LOD1_HIGH(void);
```

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

```
void GPIO_LOD_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStruct;
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = DB_LOD1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL ;
    HAL_GPIO_Init(DB_LOD1_GPIO_Port, &GPIO_InitStruct);
}
void GPIO_RST_Init(void)
```

（6）Delay_us

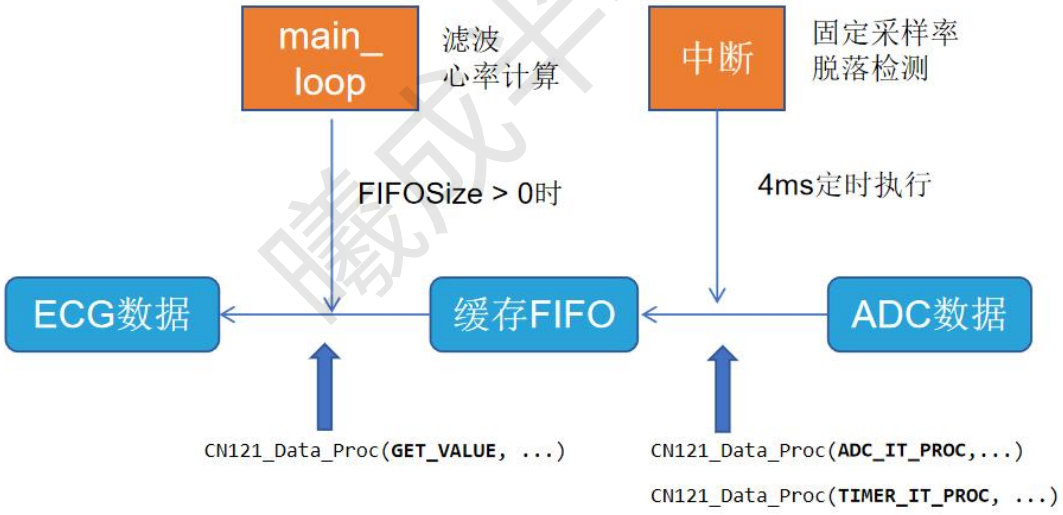
CN121_API.h 内部函数需要调用一个微秒级的延时，用于模拟 SPI 的时钟生成。目前 SDK 是使用 HAL 库生成的，如果移植，**请务必改写成适用平台的写法。 若使用软件 SPI 或者 RST 引脚，该函数内部定义需要改写，若两者都不需要则内部定义为空即可。**

```
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。

CN121_SDK 输出的 ECG 为滤波后的信号。利用 CN121_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 CN121_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。

CN121_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。

```
1.  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
2.  {
3.      int16_t adc_val_temp = (int16_t)(HAL_ADC_GetValue(&hadc1) & 0x0000ffff);
4.      CN121_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 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）**CN121_Data_Proc()**函数：使用 **TIMER_IT_PROC** 工作模式，在这个过程中，由第二个参数控制执行的功能：传参 1，只执行导联脱落；传参 2，只执行 AutoFR（自动快速恢复功能）；传参 0，同时执行导联脱落和 AutoFR。

其返回值有四种情况：

0 代表：FR 不工作，默认状态；

1 代表：配置芯片进入快速恢复状态；

2 代表：配置芯片离开快速恢复状态；

其他值代表：芯片配置失败。需要重点关注返回值为其他的情况，芯片配置失败的代表 SPI 没有配置成功。

```
1.  void TIM2_IRQHandler(void)
2.  {
```

```
    int16_t temp0 = 0;
    CN121_Data_Proc(TIMER_IT_PROC, &temp0);
    HAL_ADC_Start_IT(&hadc1);
    HAL_TIM_IRQHandler(&htim2);
}
```

（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。

```
SystemClock_Config();
ADC1_Init();
TIM2_Init();
GPIO_LOD_Init();
GPIO_RST_Init();
GPIO_SPI_Init();
USART1_UART_Init();
```

（4）配置 CN121 基本参数

然后使用 CN121_Parameter_Init 设置 CN121 的参数，当前参数代表 CN121 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。

```
CN121_Parameter_Init(HANDHELD, ADC12BIT, 3.3, 0.9);
```

（5）配置 CN121 工作模式

CN121_SDK 提供 Medical_MODE 模式和 Health_MODE 模式，通过使用 CN121_API.h 中的函数 CN121_Init(CN121_WORKMODE work_mode)配置，可为 CN121 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：GPIO 配置的模式不正确；GPIO 读写高低电平配置不正确；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(CN121_Init(Medical_MODE)) { /*printf("success!\n")*/ }
```

（6）导联脱落功能设置

然后使用 CN121_Start()函数用于开启 CN121 工作，同时开启检测导联脱落的**定时器**。CN121 芯片会检测导联是否脱落，变量 lod_status（变量值在驱动中会自动更新）表明导联的状态。

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 CN121_LOD_ON_Init(*CN121_LOD_On_MODE init_mode*)函数进行导联状态初始化。

CN121_LOD_ON_Init(*CN121_LOD_On_MODE　init_mode*)函数，有两种工作模式（若 SDK 不带心率，参数列表为空），通过传入的参数控制：

- ecgOnly：初始化后只输出心电波形数据
- realRhythm：初始化后不仅可以输出心电波形数据还可输出实时心率。

```
1.  while(1)  {
2.      CN121_Start();
3.      HAL_TIM_Base_Start_IT(&htim2);
4.      while(lod_status == LEAD_OFF)   {}
5.      CN121_LOD_ON_Init(realRhythm );  // realRhythm or ecgOnly
6.  .......
7.  }
```

（7）配置读取心电数据

CN121_Data_Proc()函数共有四种工作模式，分别是:TIMER_IT_PROC、ADC_IT_PROC 和

GET_VALUE_PROC、GET_VALUE_Rdet_PROC。前文已经用到了前两种模式。接下来重点介绍，后两种模式。

使用 GET_VALUE_PROC 模式采集心电信号 ecg 的数据。

使用 GET_VALUE_Rdet__PROC 模式采集心电信号 ecg 的数据和实时心率（驱动未提供则无此工作模式，以提供的驱动为准）。

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，CN121_Data_Proc()函数工作在 GET_VALUE_PROC 模式下的时候，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_vaue 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。

```
1.  while(lod_status == LEAD_ON)  {
2.      int16_t ecg, Rdelay;
3.      uint8_t result = CN121_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：

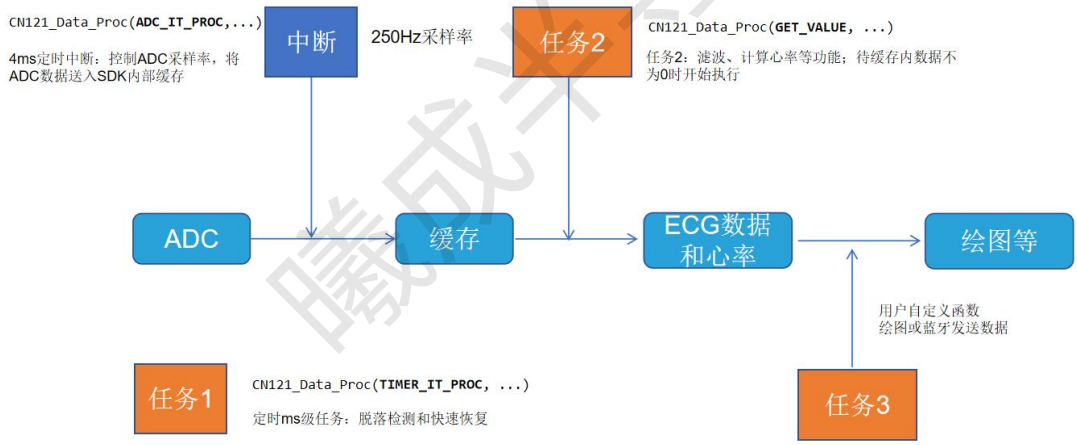
CN121_Standby()函数可以使芯片进入 **Standby 模式（具体含义见 datasheet）**。

CN121_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 CN121_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 CN121_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。

  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 CN121_Data_Proc(ADC_IT_PROC,...) 就行。

- **任务 1：**这个任务中的函数 CN121_Data_Proc(TIMER_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。

  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 CN121_Data_Proc(GET_VALUE, ...) 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。

- **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，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 CN121_Data_Proc(TIMER_IT_PROC, ...)中的不同参数来配置。

四、静态库的使用（基于 cubeIDE 平台）

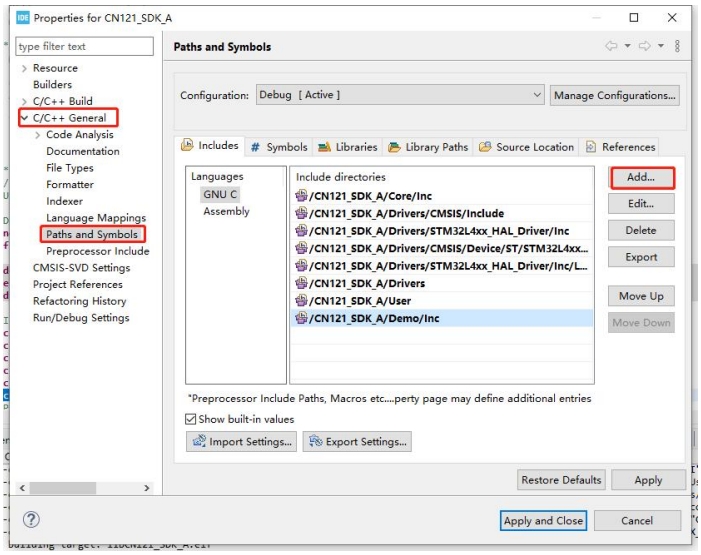
1. 将文件加入工程

直接将 CN121 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。

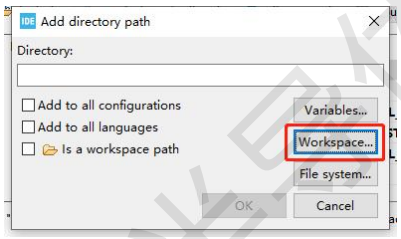
2. 头文件路径配置

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。

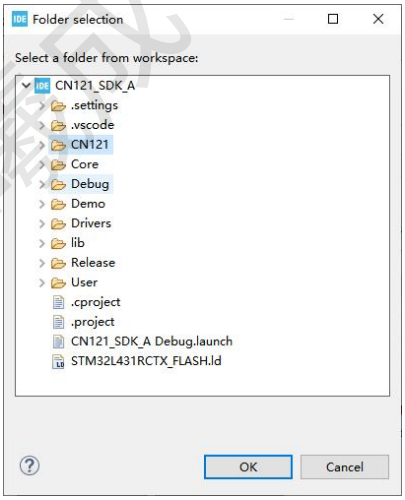
选中“C/C++ General”下的“Paths and Symbols”，点击“Add”添加路径。



点击“Workspace”。

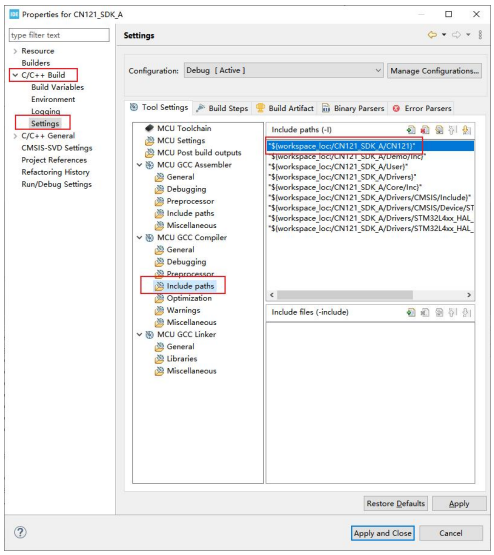


选中 CN121 文件夹并确认，点击“apply”，路径添加成功。

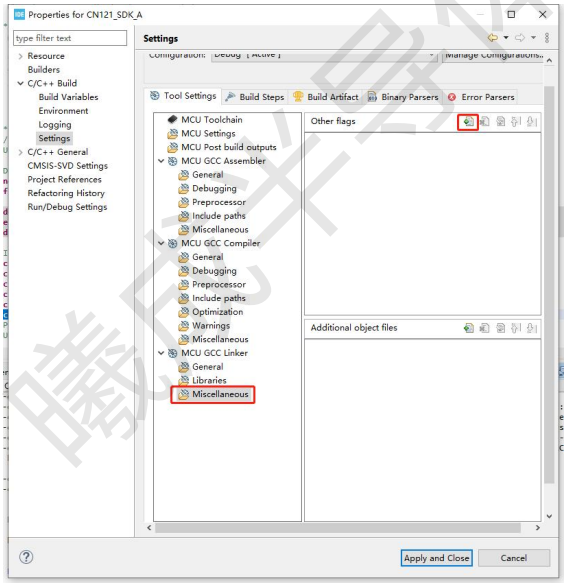


3. 静态库路径配置

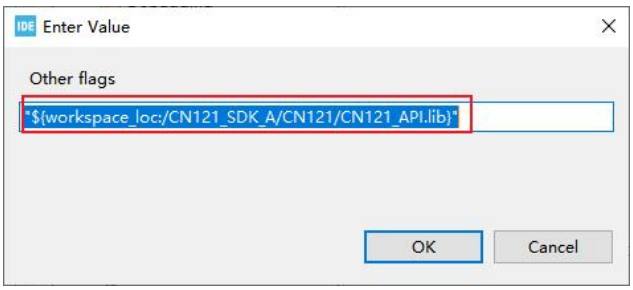
按图中步骤点击，选中 CN121 的 Inc 文件路径，双击打开后复制这个路径。



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“CN121”后面添加“/CN121_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。

五、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 CN121 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 CN121_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 CN121 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：CN121_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 CN121 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：