ZLG 致远电子

User Manual

ZM4xxSX-L 用户手册

基于 ZM4xx

UM01010101 V1.0.0 Date:2018/09/03 产品用户手册

类别	内容
关键词	ZM4xx、通用接口、移植说明、应用说明、常见问题
摘　要	描述关于 ZM4xxsx-L 系列产品软件通用接口的使用说明

修订历史

版本	日期	原因
发布 1.0.0	2018/9/3	创建文档

目 录

1. ZM4xx 系列产品软件通用接口概述

1.1 概述

ZM4xx 软件通用接口可以使 ZM4xx 系列的不同产品能够通过统一的一套软件接口操作设备，用户在更换产品时只需修改少量的应用层代码就可以直接使用，大大降低了用户重新开发软件的成本。用户使用该通用接口可以在不查看芯片手册的基础上完成模块收发数据的基本功能，大大降低了用户的学习成本和上手难度。ZM4xx 软件通用接口提供了模块收发数据，频率设置，发射功率设置和模式设置等等一些基本操作，通过这些操作就可以实现一些简单的收发功能。

1.2 结构框架

图 1: ZM4xx 软件结构框图

ZM4xx 通用软件接口结构如 图 1 所示。用户可以直接在应用层调用 Radio 层接口即可操作 ZM4xx 模块。

1.3 Demo 软件包结构

zm4xxsx-l v1.0.0 （v1.0.0 为 SDK 的版本号，以具体的版本号为准）SDK 包目录结构如 图 2 所示。

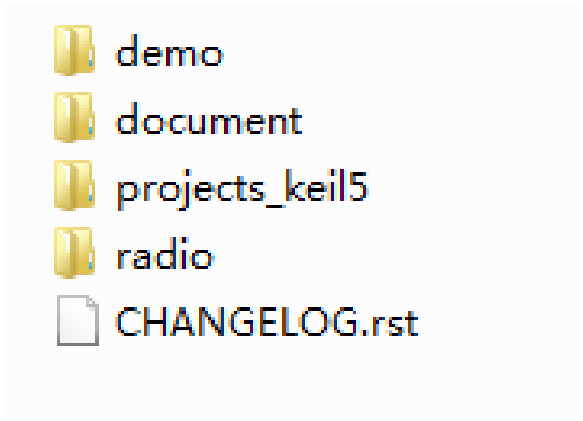

图 2: ZM4xx SDK 包目录结构图

- demo 文件夹里提供了一些简单的测试例程。
- doccument 文件夹里提供了一些文档，如 API 手册，用户手册。
- projects_keil5 文件夹是模块的 keil 工程文件，打开工程通过该文件进入。
- radio 文件夹提供了一些标准接口文件。
- CHANGELOG 文件是该模块的版本变更记录。

进入 keil 工程界面 {路径：zm4xxsx-l v1.0.0\projects_keil5\zm4xxsx-l}，打开 keil 工程。目录结构如 图 3 所示。

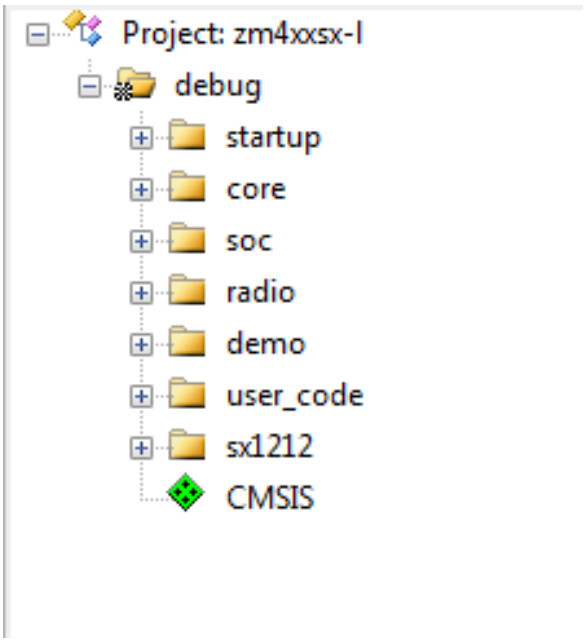

图 3: keil 工程目录结构

各个文件包含的内容如下：

- startup 和 core 文件夹包含与 MCU 相关的启动文件和内核文件，和 demo 板相关的板级初始化文件。
- soc 文件夹主要提供了 MCU 的外设驱动文件。
- radio 文件夹主要提供用户操作 ZM4xx 的标准接口文件。
- demo 文件夹提供了 demo 板的简单测试例程。

- user_code 文件夹包含主函数文件 main.c。主函数中实例初始化了无线模块，初始化成功之后运行 demo 例程。
- sx1212 文件夹目录包含了模块的驱动文件和参数配置文件。sx1212_radio_differ.h 定义了 sx1212 独有的功能。

若打开工程编译 SDK 时，出现 `radio_polling_recv_data` 问题时，请按照如 图 4 所示处理。

列表 1.1: 编译出错

```
.\soc\lpc11xx\LPC11xx.h(113): error:  #5: cannot open source input file "core_
↪cm0.h": No such file or directory
```

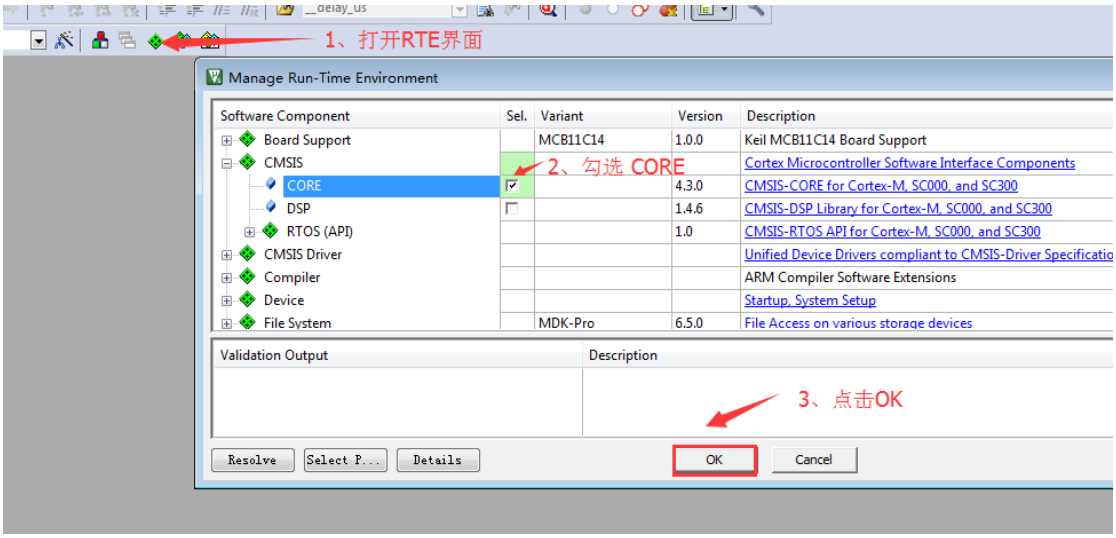

图 4: 编译出错处理

2. ZM4xx 代码移植

对于使用 ZM4XX 模块，用户可能并不是用 demo 板的 MCU 而是用自己 MCU 平台或者使用不同的编译环境，为此，用户需要将模块的驱动代码移植到自己平台上，下面介绍移植方法。

2.1 文件的移植

1. 将标准接口移植到用户的平台中，即 radio 文件夹里的所有文件。
2. 将 sx1212 驱动文件移植到用户的平台中，即 sx1212 里所有文件 {路径：zm4xxsx-l v1.0.0\projects_keil5\zm4xxsx-l\sx1212}。

2.2 文件的修改

为了适配用户的平台，只需要修改 sx1212_radio_cfg.c 文件内容即可。首先，用户需要提供以下函数。

1. 准备 SPI 驱动

ZM4xx 系列产品都是使用 SPI 接口，SPI 驱动的正常运行对 ZM4xx 的正常运行至关重要。ZM4xx 仅需要用户提供 SPI 读字节和写字节函数即可，函数格式和简单范例如 列表 2.1

所示，MCU 的主机 SPI 特性如下：

- SPI 主机采用模式 0，CPOL=0 和 CPHA=0;
- 单工通信；
- 数据长度 8 位，MSB 通信；
- SPI 速度需要小于 1M；

列表 2.1: SPI 读写函数

```
typedef struct sx1212_spi_funcs {
    /** \brief SPI 写一个字节函数 (必须提供)  */
    void    (*pfn_spi_write_byte) (uint8_t byte);

    /** \brief SPI 读一个字节函数 (必须提供) */
    uint8_t (*pfn_spi_read_byte)  (void);
} sx1212_spi_funcs_t;

// 简单范例

// SPI 发送一个字节
void spi_send_byte (uint8_t byte)
{
    //发送一个字节
}
// SPI 接受一个字节
uint8_t spi_recv_byte (void)
{
    //返回接受一个字节
}
```

注意：特别注意：用户不需要在函数内部操作 CS 片选引脚，函数内部只需单纯的发送一个字节数据或读取一字节数据。

2. 准备 GPIO 操作驱动

ZM4xxsx-l 除了 SPI 的 SCK、MISO 和 MOSI 引脚外，还有两个片选引脚，两个中断引脚，驱动中需要对这个几个引脚进行操作，因此需要用户提供这几个引脚的 GPIO 操作函数：读引脚电平函数和设置引脚电平函数。函数格式如 列表 2.2 所示。

列表 2.2: GPIO 操作函数集

```
typedef struct sx1212_gpio_funcs {

    /** \brief 数据片选引脚设置 val=0, 表示设置为低电平  val =1 表示设置为高电平 (必须提供)  */
    void    (*pfn_nss_dat_pin_set) (uint8_t val);

    /** \brief 配置片选引脚设置 val=0, 表示设置为低电平  val =1 表示设置为高电平 (必须提供) */
    void    (*pfn_nss_cfg_pin_set) (uint8_t val);

    /** \brief IRQ0 中断引脚电平读取, 返回值为电平状态, 返回 1 表示高电平, 返回 0 表示低电平 (buffer 模式必须提供) */
    uint8_t (*pfn_irq0_pin_read)   (void);

    /** \brief IRQ1 中断引脚电平读取, 返回值为电平状态, 返回 1 表示高电平, 返回 0 表示低电平 (必须提供) */
```

```
    uint8_t (*pfn_irq1_pin_read)   (void);

} sx1212_gpio_funcs_t;

// GPIO 操作函数的简单范例。

/* 复位引脚电平设置 */
void zm4xx_nss_cfg_pin_set (uint8_t val)
{
    if (val == 1) {
        //引脚输出高电平;
    } else if (val == 0) {
        //引脚输出低电平;
    }
}

/* 读 DIO0 引脚电平 */
uint8_t zm4xx_irq0_pin_read (void)
{
   //返回引脚电平，返回 1 表示高电平，返回 0 表示低电平
}
```

注意：为了能有效的控制 GPIO，用户需要自行初始化引脚配置。

3. 提供延时函数

ZM4xx 内部读写寄存器有一定的时序要求，所以需要用户提供延时函数。包括微秒延时和毫秒延时两个函数。函数格式如 列表 2.3 所示。

列表 2.3: 延时函数

```
/*
 * \brief 延时函数集
 */
typedef struct sx1212_delay_funcs {
    /** \brief ms 延时 (必须提供) */
    void (*pfn_delay_ms) (uint16_t ms);

    /** \brief us 延时 (必须提供) */
    void (*pfn_delay_us) (uint16_t us);
} sx1212_delay_funcs_t;

// 使用范例
void timer0_16_delay_ms(uint16_t ms)
{
     // ms 为延时的毫秒数
}

void timer0_16_delay_us(uint16_t us)
{
     // us 为延时的微秒数
}
```

准备好以上函数之后，用户需要将这些函数注册到驱动里面。sx1212_radio_cfg.c 文件提供了一个模板，如 列表 2.4 所示。用户只需要将上述说的三种函数对应的替换成自己的函数即可，其他地方不必修改。

列表 2.4: 延时函数

```
#include "stddef.h"
#include "timer.h"
#include "sx1212.h"
#include "zm4xx_gpio.h"

static sx1212_spi_funcs_t    __g_spi_funcs;
static sx1212_gpio_funcs_t   __g_gpio_funcs;
static sx1212_delay_funcs_t  __g_delay_funcs;
static radio_sx1212_dev_t    __g_sx1212;

#define __SYNC_LEN    4   /**< \brief 同步码长度 1 ~ 4  */

/* 同步码 */
static uint8_t __g_sync_id[4] = {0X12,0X34,0X56,0X78};
/*
 * \brief 设备信息
 */
static const radio_sx1212_info_t __g_sx1212_devinfo = {
        RF_SX1212_BITRATE_100000,          /* 传输速度 */
        RF_SX1212_MODE_PACKET,             /* 传输模式 */
        0x00,                              /* 节点地址 */
        RF_SX1212_NODE_ADDR_FILT_00,       /* 关闭地址过滤 */
        __g_sync_id,                       /* 同步码 */
        __SYNC_LEN,                        /* 同步码长度 */
};

radio_handle_t radio_zm4xx_inst_init (void)
{
    /* SPI 读写函数设置 */
    __g_spi_funcs.pfn_spi_read_byte  = spi_recv_byte;
    __g_spi_funcs.pfn_spi_write_byte = spi_send_byte;

    /* GPIO 操作函数设置 */
    __g_gpio_funcs.pfn_irq0_pin_read   = zm4xx_irq0_pin_read;
    __g_gpio_funcs.pfn_irq1_pin_read   = zm4xx_irq1_pin_read;
    __g_gpio_funcs.pfn_nss_cfg_pin_set = zm4xx_nss_cfg_pin_set;
    __g_gpio_funcs.pfn_nss_dat_pin_set = zm4xx_nss_dat_pin_set;

    /* 延时函数设置 */
    __g_delay_funcs.pfn_delay_ms = timer0_16_delay_ms;
    __g_delay_funcs.pfn_delay_us = timer0_16_delay_us;

    return radio_sx1212_init(&__g_sx1212,
                             &__g_spi_funcs,
                             &__g_gpio_funcs,
                             &__g_delay_funcs,
                             &__g_sx1212_devinfo);
}
```

3. ZM4xx 模块中断说明

对于 zm4xxsx-l 模块，有两个中断引脚，IRQ0(#5) 和 IRQ1(#6)。对于硬件上，用户需要将该引脚连接到 MCU 的 GPIO 口中。

由于驱动中将 IRQ1 配置成一接收到数据将会产生上升沿的中断功能，为此，用户可以将 IRQ1 连接对于的 GPIO 配置成上升沿中断，当模块接收到数据之后，触发中断，用户可以在中断函数里调用 radio_buf_recv (radio_handle_t handle, uint8_t *p_buf, uint8_t *p_size) 函

数去接收数据，这样就可以及时读取接收到的数据。

若用户不使用中断功能，用户可以轮询的去执行 radio_buf_recv () 函数，查看返回值是否接收到数据。

4. 操作流程

4.1 接收数据

对于接收数据模块，使用 ZM4xx 通用接口操作流程如下：

1. 获取句柄对象。

调用 sx1212_radio_cfg.c 文件的 radio_zm4xx_inst_init () 函数进行实例初始化模块，返回值为无线模块的操作句柄。

注意： 由于使用模块时，将会用到 SPI，GPIO，延时等函数，因此在实例初始化模块之前需要对所用到的 MCU 外设进行初始化。

2. 接收端进入接收模式接收数据的一方需要调用 radio_mode_set (handle, RX_MODE) 进入接收模式。

3. 接收数据 - 中断接收，配置好 IRQ1 中断引脚中断，在中断函数里调用 radio_buf_recv (radio_handle_t handle, uint8_t *p_buf, uint8_t *p_size) 接收数据

- 轮询接收，轮询接收函数在中断函数里调用 radio_buf_recv ()，判断其返回值，确定是否接收到数据。

简单的使用例程如 radio_polling_recv_data 所示。

列表 4.1: 轮询接收数据

```
void demo_zm4xx_rxdata_polling (radio_handle_t handle)
{

    uint16_t i   = 0;
    uint16_t len = 0;
    int      ret = RADIO_RET_OK;

    /* 设置为接收状态  */
    radio_mode_set(handle, RX_MODE);

    while (1) {

        /*  轮询接收  */
        ret = radio_buf_recv(handle, __g_data_buf, &len);  /*  接收数据  */
        if (ret == RADIO_RET_OK) {
            //表示接收到数据
        }
    }
}
```

4.2 发送数据

对于发送数据模块，使用 ZM4xx 通用接口操作流程如下：

1. 获取句柄对象。

调用 sx1212_radio_cfg.c 文件的 radio_zm4xx_inst_init () 函数进行实例初始化模块，返回值为无线模块的操作句柄。

2. 发送数据

调用 radio_buf_send (radio_handle_t handle, uint8_t *p_buf, uint8_t size) 函数发送数据。

注意： 若用户发送完数据之后需要接收数据，需要重新配置成接收模式。

简单的使用例程如 `radio_send_data` 所示。

列表 4.2: 发送数据

```
static uint8_t        __g_data_buf[256] = {0};

void demo_zm4xx_txdata_test (radio_handle_t handle)
{
    uint16_t  pkt_size = 10;

    for（int i = 0 ; i < pkt_size; i++ ）{
        __g_data_buf[i] = i;
    }

    /*  发送数据  */
     radio_buf_send(handle, __g_data_buf, pkt_size);
}
```

5. ZM4xxSX-L 常见使用问题总结

5.1 初始化失败，句柄返回为 NULL

这种情况很可能是 SPI 和 PIO(引脚操作) 问题。

- 1. 确认 SPI 初始化是否有问题。SPI 速率要确保在要求范围内，必须小于 1Mhz，SPI 模式为 CPOL=0 和 CPHA=0。
- 2. 确认 SPI 读写函数内部没有操作 CS 引脚；SPI 写函数内部要判断数据传输完成才能退出。
- 3. 确认 CS 引脚操作函数能否能使引脚输出高低电平。ZM4xxSX-L 比较特殊有两个片选引脚，一个为寄存器配置片选，一个为数据读写片选。

5.2 切换频率后无法接收数据

因为切换频率时内部会切换到 Standby 模式，所以修改完频率后需要重新切换到接收模式。

5.3 接收不到数据

- 1. 确认双方的配置参数是否一致，如传输模式（buffer 模式或 packet 模式）、频率、速率、前导码长度和同步 ID 等。

- 2. 确认接收端是否已进入接收模式。
- 3. 如果采用在引脚中断服务函数里接收数据，要确认中断引脚是否正确，引脚中断触发条件是否正确，ZM4xxSX-L 引脚为上升沿中断。

5.4 丢包问题

考虑到接收端处理数据需要一定时间，所以发送端应间隔一定时间再发送下一包数据，防止接收端无法及时处理数据包而导致丢失。

5.5 发送函数卡死无法返回

- 1. 因为发送函数是一个同步的函数，需要检测到 DIO0 电平拉高后即发送完成后才退出。所以 DIO0 需要配置为输入模式，使模块驱动能够检测到引脚的电平变化。
- 2. 查看是否设置的频率超出了范围，ZM433SX-L 频率 400Mhz~450Mhz，ZM470SX-L 频率 430Mhz~490Mhz。

5.6 开启地址过滤功能后无法接收数据

地址过滤功能开启之后，用户需要将发送的数据包的第一个字节设置为接收端的地址。如下图为包格式，对于数据长度，驱动已经做了处理。所以用户只需关注 (地址字节 + 消息）组成的数据包。对于接收设备来说，将接收到（地址字节 + 消息）的数据包

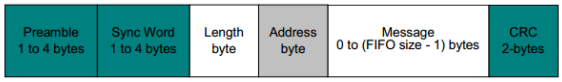

图 5: ZM4xxSX-L 数据包格式

5.7 关闭 CRC 检验后无法接收到数据

因为软件默认是 CRC 检验成功（CRC_OK 中断映射在 IRQ0）后产生接收完成中断，关闭 CRC 后该中断就无法产生。所以需要改用 Payload_ready 中断，该中断表示数据包接收完成，Payload_ready 中断映射在 IRQ1，所以 MCU 需要检测 IRQ1 来替代之前 IRQ0 中断。

5.8 发送端发送数据，接收端接收不到或者接收数据错误

确认发送端发送的包长是否超过 63byte（若开启了地址过滤，包长包括地字节）。

5.9 频率设置问题

由于频率计算需要使用多层循环嵌套进行参数枚举，效率太慢，所以使用列表的方法列出间隔 1Mhz 有限信道的频率值，有些频率无法计算到准确的频点，因此有偏移的频点没有列入表内，如 图 6，详细内容请参考 sx1212.c。当用户设置了表内没有的频点，模块将保持原有频率不变。

```
const freq_reg_param_t __g_freq_param_tab[] = {
        {400000000, 116, 42, 25},   /* 400Mhz */
        {401000000, 143, 52, 35},   /* 401Mhz */
        {402000000, 131, 48, 10},   /* 402Mhz */
        {403000000, 144, 53, 8 },   /* 403Mhz频率有偏差 */
        {404000000, 125, 46, 10},   /* 404Mhz */
        {405000000, 119, 44, 0 },   /* 405Mhz */
        {406000000, 143, 53, 10},   /* 406Mhz */
        {407000000, 143, 53, 20},   /* 407Mhz */
        {408000000, 119, 44, 25},   /* 408Mhz */
        {409000000, 143, 53, 40},   /* 409Mhz */
        {410000000, 107, 40, 0 },   /* 410Mhz */
        {411000000, 156, 58, 56},   /* 411Mhz */
        {412000000, 125, 47, 5 },   /* 412Mhz */
        {413000000, 143, 54, 5 },   /* 413Mhz */
        {414000000, 119, 45, 0 },   /* 414Mhz */
        {415000000, 143, 54, 25},   /* 415Mhz */
        {416000000, 116, 44, 5 },   /* 416Mhz */
        {417000000, 119, 45, 25},   /* 417Mhz */
        {418000000, 144, 55, 9 },   /* 418Mhz频率有偏差 */
        {419000000, 153, 58, 56},   /* 419Mhz频率有偏差 */
        {420000000, 125, 48, 0 },   /* 420Mhz */
        {421000000, 143, 55, 10},   /* 421Mhz */
        {422000000, 107, 41, 15},   /* 422Mhz */
        {423000000, 119, 46, 0 },   /* 423Mhz */
        {424000000, 125, 48, 35},   /* 424Mhz */
        {425000000, 143, 55, 50},   /* 425Mhz */
        {426000000, 119, 46, 25},   /* 426Mhz */
        {427000000, 166, 65, 2 },   /* 427Mhz频率有偏差 */
        {428000000, 143, 56, 5 },   /* 428Mhz */
        {429000000, 143, 56, 15},   /* 429Mhz */
        {430000000, 107, 42, 0 },   /* 430Mhz */
```

图 6: ZM4xxSX-L 频率参数表