User Manual

ZM4xx API 手册

基于 ZM4xx

UM01010101 V1.0.0 Date:2018/09/07

产品用户手册

类别	内容
关键词	ZM4xx、通用接口
摘　要	描述关于 ZM4xx 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	2018/9/7	创建文档

目 录

1. ZM4xx API 接口函数

1.1 驱动初始化接口

为了操作 ZM4XX 系列模块，广州致远电子股份有限公司提供了一套无线模块通用接口。在使用接口函数之前，必须先对驱动进行初始化，初始化的函数为

radio_handle_t **radio_xxxx_init**(radio_xxxx_dev_t **p_xxxx*, xxxx_spi_funcs_t **p_spi*, xxxx_gpio_funcs_t **p_gpio*, radio_xxxx_delay_t **p_delay*, radio_xxxx_info_t **p_info*)

其中，xxxx 表示模块芯片型号，ZM4xxSX-L 为 sx1212、ZM4xxSX-M 为 sx127x、ZM4xxS-M 为 si4438 和 ZM7139 为 a7139。函数原型如 列表 1.1 所示。该函数位于模块芯片驱动的头文件的末尾处，如 ZM4xxSX-L 在 sx1212.h 文件、ZM4xxSX-M 在 sx127x_lora.h(sx127x_fsk.h) 文件、ZM4xxS-M 在 si4438.h 文件和 ZM7139 在 a7139.h 文件。该函数的作用是让用户为模块驱动提供所需的 SPI 读写函数，引脚操作函数，延时函数和参数配置等，然后利用用户的配置来对模块初始化，初始化完成后返回一个句柄对象。

列表 1.1: ZM4xx 模块初始化函数

```
/**
 * \brief 驱动初始函数
 *
 * \param[in]  p_xxxx    : 设备对象
 * \param[in]  p_spi     : SPI 操作函数
 * \param[in]  p_gpio    : GPIO 操作函数
 * \param[in]  p_delay   : 延时函数
 * \param[in]  p_info    : 设备信息
 *
 * \retval  NULL        : 初始化失败
 * \retval  其它值      :  初始化成功
 */
radio_handle_t radio_xxxx_init (radio_xxxx_dev_t       *p_xxxx,
                                xxxx_spi_funcs_t  *p_spi,
                                xxxx_gpio_funcs_t *p_gpio,
                                xxxx_delay_t      *p_delay,
                                radio_xxxx_info_t      *p_info);
```

列表 1.2: SX1278 初始化接口使用示例

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

    /* GPIO 操作函数设置 */
    __g_gpio_dev.pfn_reset_pin_set = zm4xx_rst_pin_set;
    __g_gpio_dev.pfn_sel_pin_set   = zm4xx_sel_pin_set;
    __g_gpio_dev.pfn_dio0_pin_read = zm4xx_dio0_pin_read;

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

    /* 设置设备信息 */
    __g_sx127x_devinfo.p_events     = &__g_sx127x_events;
    __g_sx127x_devinfo.p_lora_param = &__g_lora_param;
```

```
    return radio_sx127x_lora_init(&__g_sx127x,
                                  &__g_spi_dev,
                                  &__g_gpio_dev,
                                  &__g_delay,
                                  &__g_sx127x_devinfo);
}
```

1.2 标准接口

int **radio_reset**(*radio_handle_t handle*)

此函数是模块复位函数，复位后需要重新初始化模块。函数原型如 列表 1.3 所示。

列表 1.3: ZM4xx 模块复位函数

```
/**
 * \brief 无线模块复位
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK            : 复位成功
 * \retval  RADIO_RET_PARAM_INVALID : 参数无效
 */
int radio_reset (radio_handle_t handle);
```

int **radio_sync_id_set**(*radio_handle_t handle*, uint8_t *p_id, uint8_t *id_len*)

此函数是设置模块当前同步 ID 和 ID 长度函数。函数原型如 列表 1.4 所示。

列表 1.4: ZM4xx 获取状态函数

```
/**
 * \brief 设置无线模块同步 ID
 *
 * \param[in]  handle   : 无线模块操作句柄
 * \param[in]  p_id     : ID 数据的指针
 * \param[in]  id_len   : ID 长度
 *
 * \retval  RADIO_RET_OK                 : 设置成功
 * \retval  RADIO_RET_PARAM_INVALID      : 参数无效
 * \retval  RADIO_RET_CFG_OUT_OF_RANGE   : ID 长度超出范围
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 模块不支持该功能
 */
int radio_sync_id_set (radio_handle_t handle, uint8_t *p_id, uint8_t id_len);
```

int **radio_sync_id_get**(*radio_handle_t handle*, uint8_t *p_id, uint8_t *p_len*)

此函数是获取模块当前同步 ID 函数。函数原型如 列表 1.5 所示。

列表 1.5: ZM4xx 获取状态函数

```
/**
 * \brief 获取无线模块同步 ID
 *
 * \param[in]  handle   : 无线模块操作句柄
 * \param[in]  p_id     : ID 数据的指针
 * \param[in]  p_len    : ID 长度
 *
 * \retval  RADIO_RET_OK                 : 获取成功
 * \retval  RADIO_RET_PARAM_INVALID      : 参数无效
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 模块不支持该功能
 */
```

```
int radio_sync_id_get (radio_handle_t handle, uint8_t *p_id, uint8_t *p_len);
```

int **radio_preamble_length_set**(*radio_handle_t handle*, uint16_t *size*)

此函数是设置前导码长度函数。函数原型如 列表 1.6 所示。

列表 1.6: 设置前导码长度函数

```
/**
 * \brief 设置无线模块前导码长度
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[in]  size      : ID 数据的指针
 *
 * \retval  RADIO_RET_OK                : 设置成功
 * \retval  RADIO_RET_PARAM_INVALID     : 参数无效
 * \retval  RADIO_RET_CFG_OUT_OF_RANGE : 前导码长度超出范围
 */
int radio_preamble_length_set (radio_handle_t handle, uint16_t size);
```

int **radio_preamble_length_get**(*radio_handle_t handle*, uint16_t **p_len*)

此函数是获取前导码长度函数。函数原型如 列表 1.7 所示。

列表 1.7: 获取前导码长度函数

```
/**
 * \brief 获取无线模块前导码长度
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[out] p_len     : 前导码长度
 *
 * \retval  RADIO_RET_OK            : 获取成功
 * \retval  RADIO_RET_PARAM_INVALID : 参数无效
 */
int radio_preamble_length_get (radio_handle_t handle, uint16_t *p_len);
```

int **radio_pa_set**(*radio_handle_t handle*, uint8_t *level*)

此函数是设置模块发射功率函数。函数原型如 列表 1.8 所示。

列表 1.8: 设置模块发射功率函数

```
/**
 * \brief 设置无线模块发射功率档位（0~7 档）
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[in]  level     : 功率档位
 * \param[out] p_power   : 实际的设置功率
 *
 * \retval  RADIO_RET_OK            : 设置成功
 * \retval  RADIO_RET_ERR           : 设置失败
 * \retval  RADIO_RET_PARAM_INVALID : 参数无效
 */
int radio_pa_set (radio_handle_t handle, uint8_t level, float *p_power);
```

int **radio_frq_set**(*radio_handle_t handle*, uint32_t *freq*)

此函数是设置模块信号频率函数，单位 Hz。函数原型如 列表 1.9 所示。

列表 1.9: 设置频率函数

```
/**
 * \brief 设置无线模块频率
```

```
 *
 * \param[in]  handle   : 无线模块操作句柄
 * \param[in]  freq     : 频率
 *
 * \retval  RADIO_RET_OK               : 设置成功
 * \retval  RADIO_RET_PARAM_INVALID    : 参数无效
 * \retval  RADIO_RET_CFG_OUT_OF_RANGE : 频率超出范围
 * \retval  RADIO_RET_FREQ_NOT_SUPPORT : 不支持该频率
 */
int radio_freq_set (radio_handle_t handle, uint32_t freq);
```

int **radio_rssi_read**(*radio_handle_t handle*, int16_t **p_rssi*)

此函数是获取模块接收信号强度函数，单位 dBm。函数原型如 列表 1.10 所示。

列表 1.10: 读信号强度函数

```
/**
 * \brief 读无线模块当前 RSSI 值
 *
 * \param[in]  handle : 无线模块操作句柄
 * \param[out] p_rssi : RSSI
 *
 * \retval  RADIO_RET_OK               : 获取成功
 * \retval  RADIO_RET_PARAM_INVALID    : 参数无效
 */
int radio_rssi_read (radio_handle_t handle, int16_t *p_rssi);
```

int **radio_mode_set**(*radio_handle_t handle*, uint8_t *mode*)

此函数是设置模块模式函数。函数原型如 列表 1.11 所示。

列表 1.11: 模式设置函数

```
/**
 * \brief 设置无线模块模式
 *
 * \param[in]  handle   : 无线模块操作句柄
 * \param[in]  mode     : 模式
 *
 * \retval  RADIO_RET_OK                 : 设置成功
 * \retval  RADIO_RET_PARAM_INVALID      : 参数无效
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块不支持该模式
 */
int radio_mode_set (radio_handle_t handle, uint8_t mode);
```

int **radio_state_get**(*radio_handle_t handle*, radio_state_t **p_state*)

此函数是获取模块当前状态函数。函数原型如 列表 1.12 所示。函数返回值类型请详见 2.2 设备运行状态说明。

列表 1.12: 获取运行状态函数

```
/**
 * \brief 获取无线模块当前状态
 *
 * \param[in]  handle  : 无线模块操作句柄
 * \param[out] p_state : 模块状态
 *
 * \retval  RADIO_RET_OK                 : 获取成功
 * \retval  RADIO_RET_PARAM_INVALID      : 参数无效
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块不支持该模式
 */
```

```
int radio_state_get (radio_handle_t handle, radio_state_t *p_state);
```

int **radio_buf_send**(*radio_handle_t handle*, uint8_t **buf*, uint8_t *size*)

此函数是模块发送数据函数。函数原型如 列表 1.13 所示。

列表 1.13: 发送数据函数

```
/**
 * \brief 无线模块发送数据
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[in]  p_buf     : 发送数据缓冲区
 * \param[in]  size      : 数据长度
 *
 * \retval  RADIO_RET_OK                : 发送成功
 * \retval  RADIO_RET_PARAM_INVALID     : 参数无效
 * \retval  RADIO_RET_CFG_OUT_OF_RANGE  : 包长度超出范围
 */
int radio_buf_send (radio_handle_t handle, uint8_t *p_buf, uint8_t size);
```

int **radio_buf_recv**(*radio_handle_t handle*, uint8_t **p_buf*, uint8_t **p_size*)

此函数是模块接收数据函数。函数原型如 列表 1.14 所示。

列表 1.14: 接收数据函数

```
/**
 * \brief 无线模块接收数据
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[in]  p_buf     : 接收数据缓冲区
 * \param[in]  size      : 接收到数据的长度
 *
 * \retval  RADIO_RET_OK             : 接收成功
 * \retval  RADIO_RET_PARAM_INVALID  : 参数无效
 * \retval  RADIO_RET_NOT_RECV_PKT   : 没有接收到数据包
 */
int radio_buf_recv (radio_handle_t handle, uint8_t *p_buf, uint8_t *p_size);
```

int **radio_ioctl**(*radio_handle_t handle*, int *cmd*, void **p_arg*)

此函数是模块控制函数，可根据命令执行相应操作。函数原型如 列表 1.15 所示。接口的命令和参数，请参考 2.4 设备控制命令说明

列表 1.15: 控制函数

```
/**
 * \brief 无线模块控制函数
 *
 * \param[in]  handle    : 无线模块操作句柄
 * \param[in]  cmd       : 控制命令
 * \param[in]  p_arg     : 该命令对应的参数
 *
 * \retval  RADIO_RET_OK                : 命令执行成功
 * \retval  RADIO_RET_ERR               : 命令执行失败
 * \retval  RADIO_RET_PARAM_INVALID     : 参数无效
 * \retval  RADIO_RET_CMD_NOT_SUPPORT   : 不支持该命令
 */
int radio_ioctl (radio_handle_t handle, int cmd, void *p_arg);
```

int **radio_dio0_irq_func**(*radio_handle_t handle*)

此函数是 ZM4xxSX-M 模块 DIO0 中断服务函数，主要处理模块的一些中断事件，需要在相应的 MCU 引脚中断服务函数里调用该函数。其它模块无该函数，不需要调用。

列表 1.16: DIO0 引脚中断服务函数

```
/**
 * \brief 无线模块 DIO0 引脚中断服务函数
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK                : 中断服务函数执行完成
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块无中断服务函数
 */
int radio_dio0_irq_func (radio_handle_t handle);
```

列表 1.17: 接口使用示例

```
void PIOINT0_IRQHandler (void)
{
    if (LPC_GPIO0->MIS & (1ul << 7)) {
        key3_int_handle();
    } else if (LPC_GPIO0->MIS & DIO0_PIN) {
        radio_dio0_irq_func(__gp_handle);
        radio_recv_int_handle();
    }

    LPC_GPIO0->IC |= (1ul << 7) | DIO0_PIN;    /*  清除中断标志  */
}
```

int **radio_dio1_irq_func**(*radio_handle_t handle*)

此函数是 ZM4xxSX-M 模块 DIO1 中断服务函数，主要处理模块的一些中断事件，需要在相应的 MCU 引脚中断服务函数里调用该函数。其它模块无该函数，不需要调用。

列表 1.18: DIO1 引脚中断服务函数

```
/**
 * \brief 无线模块 DIO1 引脚中断服务函数
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK                : 中断服务函数执行完成
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块无中断服务函数
 */
int radio_dio1_irq_func (radio_handle_t handle);
```

int **radio_dio2_irq_func**(*radio_handle_t handle*)

此函数是 ZM4xxSX-M 模块 DIO2 中断服务函数，主要处理模块的一些中断事件，需要在相应的 MCU 引脚中断服务函数里调用该函数。其它模块无该函数，不需要调用。

列表 1.19: DIO2 引脚中断服务函数

```
/**
 * \brief 无线模块 DIO2 引脚中断服务函数
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK                : 中断服务函数执行完成
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块无中断服务函数
 */
int radio_dio2_irq_func (radio_handle_t handle);
```

int **radio_dio3_irq_func**(*radio_handle_t handle*)

此函数是 ZM4xxSX-M 模块 DIO3 中断服务函数，主要处理模块的一些中断事件，需要在相应的 MCU 引脚中断服务函数里调用该函数。其它模块无该函数，不需要调用。

列表 1.20: DIO3 引脚中断服务函数

```
/**
 * \brief 无线模块 DIO3 引脚中断服务函数
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK                 : 中断服务函数执行完成
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块无中断服务函数
 */
int radio_dio3_irq_func (radio_handle_t handle);
```

int **radio_dio5_irq_func**(*radio_handle_t handle*)

此函数是 ZM4xxSX-M 模块 DIO5 中断服务函数，主要处理模块的一些中断事件，需要在相应的 MCU 引脚中断服务函数里调用该函数。其它模块无该函数，不需要调用。

列表 1.21: DIO5 引脚中断服务函数

```
/**
 * \brief 无线模块 DIO5 引脚中断服务函数
 *
 * \param[in]  handle   : 无线模块操作句柄
 *
 * \retval  RADIO_RET_OK                 : 中断服务函数执行完成
 * \retval  RADIO_RET_MODULE_NOT_SUPPORT : 该模块无中断服务函数
 */
int radio_dio5_irq_func (radio_handle_t handle);
```

2. ZM4xx 数据类型说明

ZM4xx 数据类型定义详见”radio.h” 头文件。因模块的不同，不同的部分都位于 radio_differ.h 中。下面介绍这些数据类型。

2.1 设备操作句柄说明

typedef radio_dev_t ***radio_handle_t**

radio_handle_t 是一个结构体指针类型，是标准接口的操作句柄。该结构体包含驱动提供的设备操作函数的指针和设备指针。如 表 1 和 表 2 所示。

表 1: radio_handle_t 表

成员	备注
const struct radio_drv_funcs *p_funcs	设备驱动函数指针
void *p_drv	设备指针

表 2: struct radio_drv_funcs 表

成员	备注
pfn_radio_reset	zm4xx 复位函数指针
pfn_radio_id_set	zm4xx 设置同步码函数指针
pfn_radio_id_get	zm4xx 获取同步码函数指针
pfn_radio_preamble_length_set	zm4xx 设置前导码长度函数指针
pfn_radio_preamble_length_get	zm4xx 获取前导码长度函数指针
pfn_radio_pa_set	zm4xx 设置功率函数指针
pfn_radio_frq_set	zm4xx 设置频率函数指针
pfn_radio_mode_set	zm4xx 设置模式函数指针
pfn_radio_state_get	zm4xx 获取状态函数指针
pfn_radio_rssi_read	zm4xx 读取 RSSI 函数指针
pfn_radio_buf_send	zm4xx 发送数据函数指针
pfn_radio_buf_recv	zm4xx 接收数据函数指针
pfn_radio_ioctl	zm4xx 控制函数指针

2.2 设备运行状态说明

enum `radio_state`

`radio_state` 是一个枚举数据类型。该枚举类型定义了模块运行的状态。如 表 3 所示。

表 3: radio_state_t 表

成员	备注
IDLE_ST	空闲状态
SLEEP_ST	睡眠状态
RX_RUNING_ST	接收状态
TX_RUNING_ST	发送状态
AUTO_WAKE_SLEEP_ST	自动唤醒状态
CAD_ST	CAD 状态

注意：ZM4xxSX-L 和 ZM4xxSX-M 无 AUTO_WAKE_SLEEP_ST 状态；
ZM4xxSX-L、ZM4xxS-M 和 ZM7139 无 CAD_ST 状态；

2.3 设备模式说明

用户可设置模块的当前运行模式。如 列表 2.1 所示。

列表 2.1: 设备运行模式

```
/* mode */
#define SLEEP_MODE                      0x00
#define STANDBY_MODE                    0x01
#define TX_MODE                         0x02
```

```
#define RX_MODE                         0x03
```

2.4 设备控制命令说明

设备控制命令为用户提供不同模块间差异化操作，命令作为*radio_ioctl()* 的 cmd 参数。如 列表 2.2 所示。

列表 2.2: 设备控制命令

```
/* 无线模块控制命令 */
#define NODE_ADDR_SET                   0x01 /* 设置匹配地址，ZM4xxSX-M 不支持 */
#define NODE_ADDR_GET                   0x02 /* 获取匹配地址，ZM4xxSX-M 不支持 */
#define NODE_ADDR_FILT_SET              0x03 /* 地址过滤模式选择，仅 ZM4xxSX-L 支
持 */
#define AUTO_WAKE_SLEEP_SET             0x04 /* 自动唤醒模式开关，仅 ZM4xxS-M 和
ZM7139 支持 */
#define AUTO_WAKE_SLEEP_TIME_SET        0x05 /* 自动唤醒时间设置，仅 ZM4xxS-M 和
ZM7139 支持 */
#define CAD_START                       0x06 /* 开启 CAD，仅 ZM4xxSX-M 支持 */
#define SYMB_TIMEOUT_SET                0x07
#define BATTERY_VOLTAGE_GET             0x08 /* 获取电量电压 */
```

列表 2.3: 使用示例

```
radio_ioctl(handle, CAD_START, NULL); /* 命令无参数，所以 p_arg 为 NULL */
radio_ioctl(handle, NODE_ADDR_FILT_SET, NODE_ADDR_FILT_01);
```

2.4.1 NODE_ADDR_FILT_SET 命令参数说明

设置设备地址匹配方式，仅 ZM4xxSX-L 支持。如 列表 2.4 所示。

列表 2.4: NODE_ADDR_FILT_SET 命令参数

```
/* 节点地址过滤选项 */
#define NODE_ADDR_FILT_00               0x00 /* 不过滤 */
#define NODE_ADDR_FILT_01               0x01 /* 只匹配 NODE_ADDR */
#define NODE_ADDR_FILT_10               0x02 /* 只匹配 0x00 和 NODE_ADDR */
#define NODE_ADDR_FILT_11               0x03 /* 只匹配 0x00、NODE_ADDR 和 0xFF
↪*/
```

列表 2.5: 使用示例

```
radio_ioctl(handle, NODE_ADDR_FILT_SET, NODE_ADDR_FILT_00);
```

2.4.2 AUTO_WAKE_SLEEP_SET 命令参数说明

设置设备自动唤醒功能是否开启，仅 ZM4xxS-M 和 ZM7139 支持。如 列表 2.6 所示。

列表 2.6: AUTO_WAKE_SLEEP_SET 命令参数

```
/* 自动唤醒功能选项 */
#define AUTO_WAKE_SLEEP_OFF             (void *)0x00
#define AUTO_WAKE_SLEEP_ON              (void *)0x01
```

列表 2.7: 使用示例

```
radio_ioctl(handle, AUTO_WAKE_SLEEP_SET, AUTO_WAKE_SLEEP_ON);
```

索引