User Manual

ZM4xx 用户手册

基于 ZM4xx

UM01010101 V1.0.0 Date:2018/07/09

产品用户手册

类别	内容
关键词	ZM4xx、通用接口、应用说明
摘　要	描述关于 ZM4xx 系列产品软件通用接口的使用说明

广州致远电子有限公司

修订历史

版本	日期	原因
发布 1.0.0	2018/7/9	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，同时也因为不同产品硬件上的不同提供了一些差异化的操作，如 ZM4xxS-M 和 ZM7139 所独有的自动睡眠和唤醒时间设置，而 ZM4xxSX-L 和 ZM4xxSX-M 是没有的。

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

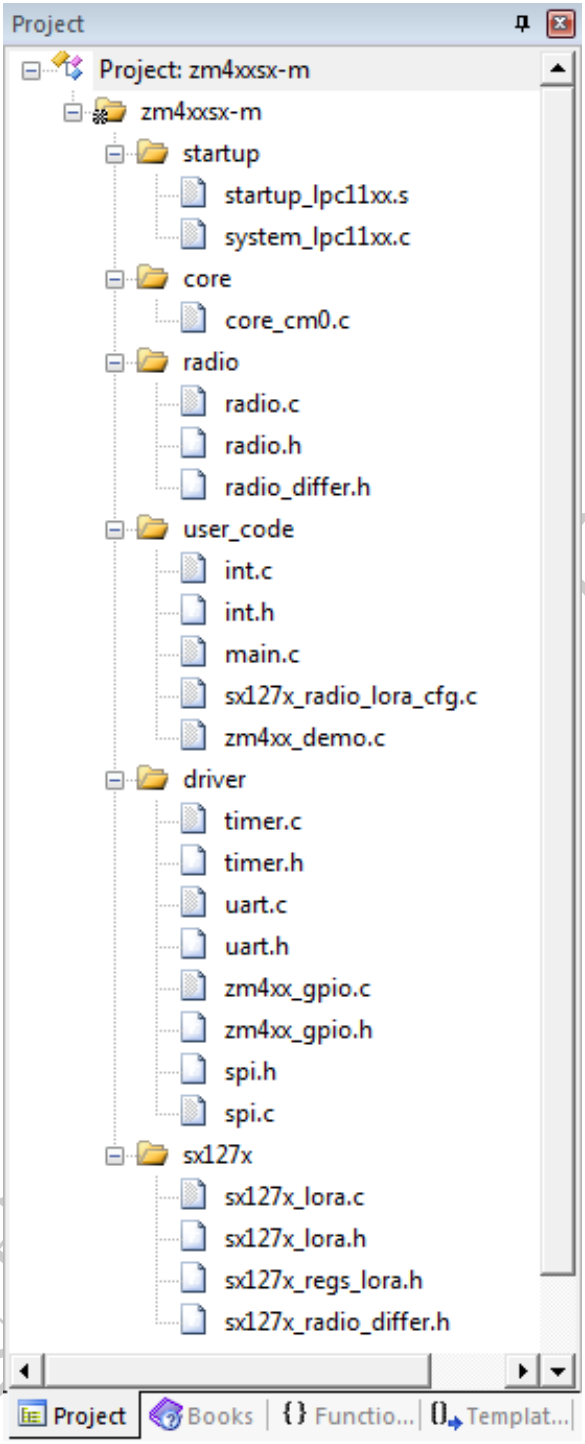

图 2: ZM4xx Demo 软件包目录图

ZM4xx Demo 软件包目录结构如 图 2 所示。

- startup 和 core 文件夹包含与 MCU 相关的启动文件和内核文件。
- radio 文件夹包含供用户操作 ZM4xx 的标准接口文件。radio_differ.h 文件是用于包含不同模块独有的功能。
- user_code 文件夹包含主函数文件 main.c，主函数初始化一些 Demo 及模块需要使用的

外设，初始化完成后进入 Demo。

- 中断服务函数 int.c，引脚中断服务函数都统一放到该文件来调用。
- sx127x_radio_cfg.c 是用户需要向驱动提供的配置参数、SPI 操作函数和 GPIO 操作函数。
- driver 目录包含了 MCU 的相关外设驱动。
- sx127x 目录包含了模块的驱动文件。sx127x_radio_differ.h 定义了 sx127x 独有的功能。

注意： 不同模块 sx127x_radio_cfg.c 与 sx127x 目录名和文件名会不同。上文仅以 ZM4xxSX-M 模块为例。

2. ZM4xx 代码移植

用户如果使用自己的 MCU 平台，需要将 xxxx_radio_cfg.c 文件，radio 目录和模块驱动目录移植过去并修改 xxxx_radio_cfg.c 即可使用。

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，函数格式如 `spi_driver_func_def` 所示，用户不需要在函数内部操作 CS 片选引脚，片选引脚在下面的 GPIO 操作单独提供。

列表 2.1: SPI 读写函数

```
void spi_send_byte (uint8_t byte);

uint8_t spi_recv_byte (void);
```

2. 准备 GPIO 操作驱动

ZM4xx 除了 SPI 的 SCK、MISO 和 MOSI 引脚外，需要对其它的引脚提供操作函数，如复位引脚电平设置，片选引脚电平设置等等，函数格式如 `gpio_ctrl_func_def` 所示。由于不同模块可能需要提供更多的引脚操作函数，详细请查看相关芯片驱动的头文件中 xxxx_gpio_funcs_t 结构体的成员。

列表 2.2: GPIO 操作函数

```
/* 复位引脚电平设置 */
void zm4xx_rst_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;
    }
}

/* 片选引脚电平设置 */
void zm4xx_sel_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_dio0_pin_read (void)
{
    return ((LPC_GPIO0->DATA & DIO0_PIN) != 0);
}
```

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 delay_func_def 所示。

列表 2.3: 延时函数

```
void timer0_16_delay_ms(uint16_t ms);

void timer0_16_delay_us(uint16_t us);
```

4. 函数注册

将上述的函数通过设备初始化函数 radio_xxxx_init() 注册到到设备内部供驱动使用。下面是 ZM4xxSX-M 函数注册的示例，如 radio_init_func_def 所示。

列表 2.4: 延时函数

```
static sx127x_spi_funcs_t   __g_spi_dev;
static sx127x_gpio_funcs_t  __g_gpio_dev;
static sx127x_timer_funcs_t __g_delay;
static radio_sx127x_dev_t   __g_sx127x;
static lora_radio_events_t  __g_event;

radio_handle_t radio_zm4xxsx_m_inst_init (void)
{
    /* 用户提供的 SPI 读写函数 */
    __g_spi_dev.pfn_spi_read_byte  = spi_recv_byte;
    __g_spi_dev.pfn_spi_write_byte = spi_send_byte;

    /* 用户提供的 GPIO 操作函数 */
    __g_gpio_dev.pfn_reset_pin_set = zm4xx_rst_pin_set;
    __g_gpio_dev.pfn_sel_pin_set   = zm4xx_sel_pin_set;

    /* 用户提供的延时函数 */
    __g_delay.pfn_delay_ms = timer0_16_delay_ms;
    __g_delay.pfn_delay_us = timer0_16_delay_us;

    /* 用户提供的回调函数 */
    __g_sx127x_events.pfn_rx_done_cb             = __rx_done_call_func;
    __g_sx127x_events.pfn_cad_done_cb            = __cad_done_call_func;
    __g_sx127x_events.pfn_rx_timeout_cb          = __rx_timeout_call_func;
    __g_sx127x_events.pfn_rx_error_cb            = __rx_crc_err_call_func;
    __g_sx127x_events.pfn_fhss_change_channel_cb = __fhss_change_channel_call_
↪func;

    __g_sx127x_cfg.p_events     = &__g_sx127x_events;
    __g_sx127x_cfg.p_lora_param = &__g_lora_param;

    /* 将上述的配置注册进模块驱动供其使用 */
    return radio_sx127x_init(&__g_sx127x,
```

```
                                &__g_spi_dev,
                                &__g_gpio_dev,
                                &__g_delay,
                                &__g_sx127x_cfg);
}
```

5. 中断调用

该项仅针对 ZM4xxSX-M 模块，其它模块请忽略。ZM4xxSX-M 有 DIO0、DIO1、DIO2、DIO3 和 DIO5 五个引脚中断服务函数，如 radio_io_int_serve_func_def , 这些中断服务函数需要放在引脚中断里面执行，如 radio_io_int_exec_func_def ， 触发条件为上升沿中断。中断处理完成后需要清中断标志。

列表 2.5: DIO 中断服务函数

```
int radio_dio0_irq_func (radio_handle_t handle);

int radio_dio1_irq_func (radio_handle_t handle);

int radio_dio2_irq_func (radio_handle_t handle);

int radio_dio3_irq_func (radio_handle_t handle);

int radio_dio5_irq_func (radio_handle_t handle);
```

列表 2.6: 引脚中断处理

```
void PIOINT2_IRQHandler(void)
{
    if (LPC_GPIO2->MIS & DIO1_PIN) {
        radio_dio1_irq_func (__gp_handle);
    } else if (LPC_GPIO2->MIS & DIO2_PIN) {
        radio_dio2_irq_func (__gp_handle);
    } else if (LPC_GPIO3->MIS & DIO3_PIN) {
        radio_dio3_irq_func (__gp_handle);
    }

    LPC_GPIO2->IC |= DIO1_PIN | DIO2_PIN | DIO3_PIN;     /*  清除中断标志  */
}
```

注意： 用户在模块初始化前需要先初始化 SPI，GPIO 和定时器。

3. 操作流程

使用 ZM4xx 通用接口操作设备分为以下几步：

1. 获取句柄对象。

需要先调用驱动函数中的 radio_handle_t radio_xxxx_inst_init (void) 初始化并返回一个句柄对象，xxxx 表示不同的模块。

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. 发送端发送数据。

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

4. 接收端处理数据。

判断是否产生接收中断，接收中断触发条件参考 表 1 ，在中断服务函数里调用 radio_buf_recv (radio_handle_t handle, uint8_t *p_buf, uint8_t *p_size) 接收数据。

表 1: 接收中断触发条件表

型号	触发条件
ZM4xxSX-L	上升沿中断
ZM4xxSX-M	上升沿中断
ZM4xxS-M	下降沿中断
ZM7139	下降沿中断

5. 重新进入接收模式。

对于不具备连续接收模式功能的模块（仅 ZM4xxSX-M 支持），需要在接收到一包数据之后重新调用函数 radio_mode_set (handle, RX_MODE) 函数进入接收模式，否则无法接收。

4. ZM4xx 标准接口使用范例

本文将 ZM4xxSX-M 模块的操作实现作为使用范例，其它模块相应的 DEMO。本范例实现了 ZM4xxSX-M 模块的初始化，频率设置，功率设置，模式设置和收发数据等基本功能。

1. 定义驱动初始化接口所需要的变量及初始化，需要初始化 SPI 操作函数，GPIO 操作函数，延时函数和配置信息。如 列表 4.1 所示。

列表 4.1: 驱动初始化

```
static sx127x_spi_funcs_t       __g_spi_dev;
static sx127x_gpio_funcs_t      __g_gpio_dev;
static sx127x_timer_funcs_t     __g_delay;
static radio_sx127x_dev_t       __g_sx127x;
static sx127x_config_t          __g_sx127x_cfg;
static sx127x_lora_events_t     __g_sx127x_events;

/*
 * Lora 参数配置
 */
static sx127x_lora_param_set_t __g_lora_param =
{
    473000000,          /* 频率 */
    20,                 /* 发射功率 */
    8,                  /* 带宽 [0: 7.8 kHz, 1: 10.4 kHz, 2: 15.6 kHz, 3: 20.8 kHz,
↪ 4: 31.2 kHz,
                           5: 41.6 kHz, 6: 62.5 kHz, 7: 125 kHz, 8: 250 kHz,␣
↪ 9: 500 kHz, other: Reserved] */
    7,                  /* 扩频因子 [6: 64, 7: 128, 8: 256, 9: 512, 10: 1024, 11:␣
↪2048, 12: 4096  chips] */
    1,                  /* 编码率   [1: 4/5, 2: 4/6, 3: 4/7, 4: 4/8] */
    true,               /* CRC 检验  [0: 关闭, 1: 开启] */
    false,              /* 报头模式 [0: 显式报头模式, 1: 隐式报头模式] 注: SF 为 6 时只
能使用隐式报头 */
    0,                  /* 接收模式 [0: Continuous, 1 Single] */
    0,                  /* 跳频     [0: 关闭, 1: 开启] */
    4,                  /* 跳频周期（symbols） */
    100,                /* 发送超时 */
```

```
    100,            /* 接收超时 */
    128,            /* 负载长度 */
    8               /* 前导码长度 */
};


radio_handle_t radio_zm4xxsx_m_inst_init (void)
{
    __g_spi_dev.pfn_spi_read_byte  = spi_recv_byte;
    __g_spi_dev.pfn_spi_write_byte = spi_send_byte;

    __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;

    __g_delay.pfn_delay_ms = timer0_16_delay_ms;
    __g_delay.pfn_delay_us = timer0_16_delay_us;

    __g_sx127x_events.pfn_rx_done_cb             = __rx_done_call_func;
    __g_sx127x_events.pfn_cad_done_cb            = __cad_done_call_func;
    __g_sx127x_events.pfn_rx_timeout_cb          = __rx_timeout_call_func;
    __g_sx127x_events.pfn_rx_error_cb            = __rx_crc_err_call_func;
    __g_sx127x_events.pfn_fhss_change_channel_cb = __fhss_change_channel_call_
↪func;

    __g_sx127x_cfg.p_events     = &__g_sx127x_events;
    __g_sx127x_cfg.p_lora_param = &__g_lora_param;

    return radio_sx127x_init(&__g_sx127x,
                             &__g_spi_dev,
                             &__g_gpio_dev,
                             &__g_delay,
                             &__g_sx127x_cfg);
}
```

2. 初始化设备并获取句柄对象，并将得到的句柄传入 demo 中使用。如 dev_init_and_func_def 所示。

列表 4.2: 初始模块并获取返回句柄

```
int main (void)
{
    radio_handle_t __gp_handle;

    /* 系统初始化；晶体 11.0592MHz   */
    SystemInit();
    LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 6);

    zm4xx_pin_init(); /* 模块引脚初始化（SPI 引脚除外） */
    spi_init();       /* 模块 SPI 初始化 */
    uart_init(115200, 256, NULL);
    timer0_16_init();

    __gp_handle = radio_sx1212_inst_init();
    if (__gp_handle == NULL) { /* 初始化失败 */
        return -1;
    }

    demo_std_radio_entry(__gp_handle);

    return 0;
}
```

3. 操作设备

列表 4.3: 初始模块并获取返回句柄

```
/**
 * \brief  无线模块接收中断服务函数
 */
void radio_recv_int_handle (void)
{
    int      ret = 0;
    uint8_t  len;

    ret = radio_buf_recv(__gp_handle, g_data_buf, &len);  /*  接收数据  */
    if (ret == 0) {
        __g_rcv_len += len;
    }
}

void PIOINT0_IRQHandler (void)
{
    if (LPC_GPIO0->MIS & (1ul << 7)) {
        key3_int_handle();
    } else if (LPC_GPIO0->MIS & DIO0_PIN) {
        __sx127x_on_dio0_irq(__gp_handle->p_drv);
        radio_recv_int_handle();
    }

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

void demo_std_radio_entry (radio_handle_t handle)
{
    uint8_t  pkt_size = 0;
    uint16_t i = 0;
    uint32_t j;

    __gp_handle = handle;

    /* 初始化 LED */
    LED_INIT();

    /* 初始化 UART  */
    uart_init(115200, 256, NULL);

    /* 初始按钮 S3 中断; 对应功率  */
    __key3_int();

    /* 初始按钮 S2 中断 */
    __key2_int();

    //radio_frq_set(handle, FREQ);

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

    /*  初始化无线模块中断  */
    radio_int_init();

    __led_show(0x18);

    while (1) {
        /*  接收  */
        if (__g_recv_byte > 0) {
```

```
            /*  把数据从串口发送出去  */
            for (i = 0; i < __g_recv_byte; i++) {
                uart_byte_send(g_data_buf[i]);
            }

            __g_recv_byte_all += __g_recv_byte;
            /*  收到数据 LED 显示计数加 1  */
            __g_recv_pkt_cnt++;
            __led_show(__g_recv_pkt_cnt % 128);

            __g_recv_byte = 0;
            radio_mode_set(handle, RX_MODE); /* 重新进入接收模式 */
        }

        /*  发送  */
        if (__g_send_byte > 0) {
            NVIC_DisableIRQ(UART_IRQn);          /* UART 中断      */
            //__disable_key_irq();               /* 关闭按键中断 */

            __led_show(0x80);

            radio_buf_send(handle, g_data_buf, __g_send_byte);       /*  发送数
据  */

            radio_mode_set(handle, RX_MODE);
            __g_send_pkts++;
            __led_show(__g_recv_pkt_cnt % 128);
            __g_send_byte = 0;
            NVIC_EnableIRQ(UART_IRQn);           /* 使能 UART 中断 */
            //__enable_key_irq();                /* 开启按键中断 */
        }

        /* 从串口获取要发送的数据包 */
        if (uart_recv_pkt(g_data_buf, &pkt_size) == 0) {
            __g_send_byte = pkt_size;
            __g_uart_byte += pkt_size;
        }
    }
}
```

销售与服务网络

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 7 栋 2 楼
邮编：510660
网址：www.zlg.cn

全国服务电话：400-888-4005

全国销售与服务电话：400-888-4005

销售与服务网络：

广州总公司

广州市天河区车陂路黄洲工业区 7 栋 2 楼
电话：020-28267893

上海分公司

上海市北京东路 668 号科技京城东楼 12E 室
电话：021-53865720

北京分公司

北京市海淀区紫金数码园 3 号楼（东华合创大厦）8 层 0802 室
电话：010-62536178

深圳分公司

深圳市福田区深南中路 2072 号电子大厦 12 楼 1203 室
电话：0755-83780058

武汉分公司

武汉市武昌区武珞路 282 号思特大厦 807 室
电话：027-87168497-613

南京分公司

南京市秦淮区汉中路 27 号友谊广场 17 层 F、G 区
电话：025-68123936

杭州分公司

杭州市天目山路 217 号江南电子大厦 502 室
电话：0571-86483297

成都分公司

成都市一环路南二段 1 号数码科技大厦 403 室
电话：028-85439836-805

郑州分公司

郑州市东大街与紫荆山路交叉口紫燕华庭 B 座 2 单元 1406 室
电话：0371-66868897

重庆分公司

重庆市九龙坡区石桥铺科园一路二号大西洋国际大厦（百脑会）2705 室
电话：023-68797619

西安办事处

西安市长安北路 54 号太平洋大厦 1201 室
电话：029-87881295

天津办事处

天津市河东区津塘路与十一经路交口鼎泰大厦 1004
电话：022-24216606

青岛办事处

山东省青岛市李沧区青山路 689 号宝龙公寓 3 号楼 311 室
电话：0532-58879795

请您用以上方式联系我们，我们会为您安排样机现场演示，感谢您对我公司产品的关注！