```
/*************************************************************
 * bc7215_lib_config.h中配置项                                *
 * TX_HW_FLOW_CONTROL，指串口数据发送的流控制，如果系统支持，如 *
 * 电脑和STM32等很多单片机上，则建议尽量采用硬件流控制，配合发送*
 * FIFO，可以完成串口的非阻塞式后台发送，程序只需将数据写入发送 *
 * 缓冲区，否则，则在串口发送期间，每发送一个字节之前驱动库必须 *
 * 检查BUSY引脚的状态，因为红外信号发送时间可能会较长，约几十到 *
 * 上百ms，此期间可能造成程序的阻塞。                          *
 * 如果没有串口发射流控的硬件，但用户自己通过底层软件实现流控的 *
 * 功能并提供FIFO缓冲区，确保每个字节发送到串口数据线上时，BUSY *
 * 为低电平，则对BC7215驱动库而言，也等同于具有硬件流控，可选择 *
 * 此项。
 * USE_UART_INTERRUPT，指串口的接收，是否由用户通过串口中断来  *
 * 处理，还是系统会自动将所接收数据写入缓冲区，用户通过读取缓冲 *
 * 区获取数据。                                               *
 *************************************************************/

#include "bc7215.h"               // 包含BC7215驱动库，提供基本操作
#include "bc7215_ac_lib.h"        // 包含空调码库头文件，实现空调控制

/*********************************************************
 *                BC7215A驱动库所需接口函数               *
 *********************************************************/
// MOD引脚置高函数，功能为设置BC7215A芯片MOD引脚为高电平
void set_mod_high(void) { MOD_PIN = 1; }

// MOD引脚置低函数，功能为设置BC7215A芯片MOD引脚为低电平
void set_mod_low(void) { MOD_PIN = 0; }

// 读取MOD引脚状态函数，返回0或1
uint8_t read_mod(void) { return MOD_PIN; }

/************* 此部分仅需在不使用发送硬件流控制时使用 ***********************************
 * // 读取BUSY引脚状态函数，返回0或1                                                  *
 * uint8_t read_busy(void)                                                          *
 * {                                                                                *
 *     return BUSY_PIN;                                                             *
 * }                                                                                *
 *                                                                                  *
 * // UART单字节发送函数                                                           *
 * void uart_send_1_byte(uint8_t data)                                              *
 * {                                                                                *
 *     UART_TXD = data;    // 待发数据写入UART发送寄存器                            *
 *     while (!UART_TX_COMPLETE);    // 等待数据完全发送完成，注意不是发送寄存器为空，     *
 *                                  // 有的MCU数据开始发送即显示发送寄存器为空，但数据     *
 *                                 // 尚未完全发出                                    *
 * }                                                                                *
 ************************************************************************************/

/******** 此部分仅需在使用发送硬件流控制时使用 ************************************
 * // UART单字节发送函数                                                          *
 * void uart_send_1_byte(uint8_t data)                                             *
 * {                                                                               *
 *     while (SEND_FIFO_IS_FULL);  // 如果发送FIFO已满，则等待                     *
 *     write_to_send_fifo(data);    // 将待发数据写入串口发送FIFO                  *
 * }                                                                               *
 ***********************************************************************************/

/******** 此部分仅需在使用中断处理串口接收时使用 ****************
 * // UART中断使能函数                                        *
 * void enable_rx_int(void)                                   *
 * {                                                          *
 *     RXIE = 1;                                              *
 * }                                                          *
 *                                                            *
 * // UART中断禁止函数                                        *
 * void disable_rx_int(void)                                  *
 * {                                                          *
 *     RXIE = 0;                                              *
 * }                                                          *
 *                                                            *
 * // UART接收中断处理函数                                    *
 * void uart_rx_isr(void)                                     *
```

```
 * {                                                                *
 *     uint8_t data;                                                *
 *     data = UART_RXD;        // 读取所收到数据                    *
 *     bc7215_process_uart_data(data); // 调用BC7215驱动库处理      *
 *       // bc7215_process_uart_data()调用后，将影响查询函数        *
 *       // bc7215_data_ready(), bc7215_format_ready(),            *
 *       // bc7215_cmd_completed(), bc7215_is_busy() 的返回结果    *
 * }                                                                *
 ******************************************************************/

/************ 此部分仅在当使用系统串口接收缓冲区时使用 **************
 * // 定期轮询串口接收缓冲区或者由串口接收事件驱动                  *
 * void check_rx_buffer(void)                                        *
 * {                                                                 *
 *    uint8_t data;                                                  *
 *    while (RX_BUFFER_NOT_EMPTY)        // 读取直至接收缓冲区为空    *
 *    {                                                              *
 *        data = read_from_rx_buffer();    // 读出一个字节           *
 *        bc7215_process_uart_data(data);    // 调用BC7215驱动库处理 *
 *          // bc7215_process_uart_data()调用后，将影响查询函数     *
 *          // bc7215_data_ready(), bc7215_format_ready(),          *
 *          // bc7215_cmd_completed(), bc7215_is_busy() 的返回结果  *
 *    }                                                              *
 * }                                                                 *
 ******************************************************************/

/****************** 以上为硬件相关代码 ******************************
------------------------------------------------------------------------
****************** 以下代码与硬件无关 ******************************/

/************************************************************
 *                   空调码库所需变量定义                    *
 ************************************************************/
// 当不确定所用空调型号（数据长度）时，使用最大数据包尺寸作为输入缓存
// 通常采样仅需要1个数据包和1个格式包缓存，但有多段数据格式，可能信号会
// 被分为三段，这里取4保留一定余量
bc7215DataMaxPkt_t IrRxData[4];
bc7215FormatPkt_t  IrRxFormat[4];
uint8_t            IrStatus[4];
// 这个变量主要用作多段格式的采样数据初始化使用
bc7215CombinedMsg_t IrRxMsg[4];
// 初始化使用的状态字
uint8_t IrStatus;
// 待发射的数据
const bc7215DataVarPkt_t* DataToSend;

/************************************************************
 *                   主程序                                  *
 ************************************************************/
main()
{
    init_uart();        // 设置串口为波特率19200, 8个数据位，无奇偶校验，2个停止位

    // 设置BC7215驱动库
    bc7215_config_mod_set_high_funct(set_mod_high);        // 配置BC7215驱动库MOD置高函数
    bc7215_config_mod_set_low_funct(set_mod_low);          // 配置BC7215驱动库MOD置低函数
    bc7215_config_read_mod_funct(read_mod);                // 配置BC7215驱动库MOD读取函数
    bc7215_config_uart_send_byte(send_1_byte);             // 配置BC7215驱动库串口单字节发送函数

    /************ 仅在不使用硬件发送流控制时使用 ****************************************************
     *    bc7215_config_read_busy_funct(read_busy);            // 配置BC7215驱动库BUSY读取函数       *
     ************************************************************************************************/

    /************ 仅在使用中断处理串口接收时使用 ****************************************************
     *    bc7215_config_uart_int_en_funct(enable_rx_int);        // 配置BC7215驱动库串口中断使能函数 *
     *    bc7215_config_uart_int_dis_funct(disable_rx_int);    // 配置BC7215驱动库串口中断禁止函数   *
     ************************************************************************************************/

    bc7215_set_tx();        // 设置BC7215A上电处于发射模式

    ac_lib_init();        // 初始化空调码库

    // 初始化已完成，进入空调控制或解析状态
    bc7215_load_format(bc7215_ac_get_base_fmt());        // 加载基础格式信息到BC7215A芯片，
                                                         // 因前面使用的是带格式的初始化方式，
```

```
                                                        // 故可使用get_base_fmt函数
                                                        // 格式信息如果不变可以不用每次发射时重复加载
    while (1)        // 无限循环
    {
        working_mode_select();          // 可能的允许用户选择工作模式的操作
        if (WorkingMode_changed)        // 如果工作模式发生了改变
        {
            if (WorkingMode == AC_PARSING)        // 如果模式切换为解析模式
            {
                bc7215_set_rx();              // 将芯片切换为接收模式
                delay_100ms();                // 确保切换完成
                bc7215_set_rx_mode(0);        // 设置为简单模式
            }
            else if (WorkingMode == AC_CONTROL)        // 如果工作模式从解析模式切换为控制模式
            {
                bc7215_set_tx();        // 将芯片切换为发射模式
                delay_100ms();          // 确保转换完成
                IrRxData[0] = EEPROM_read_data();        // 因解析模式会替换基础数据包，故最好使用初始化数据重新加载
                IrRxFormat[0] = EEPROM_read_format();
                IrStatus = IrRxFormat[0].signature.bits.sig;
                bc7215_ac_replace_base(IrStatus, (bc7215DataVarPkt_t*)&IrRxData[0]);
                bc7215_load_format(bc7215_ac_get_base_fmt());        // 加载基础格式包到芯片
            }
        }
        switch (WorkingMode)
        {
        case AC_CONTROL:                                        // 空调控制模式
            get_new_setting();                                  // 用户设置新的temp, mode, fan, key值
            DataToSend = bc7215_ac_set(temp, mode, fan, key);        // 获取新指令的数据
            /*************************************************************************
             * 如果开关机，则调用 bc7215_ac_on()或bc7215_ac_off()函数，同样返回数据指针，处理    *
             * 方法相同。开机指令如返回指针为NULL，标明无需专用开机指令                        *
             *************************************************************************/
            UsingSpecialFormat = 0;
            if (DataToSend->bitLen == 0)        // 如果bitLen为0，表明返回实际为一个bc7215CombinedMsg_t*指针
            {
                bc7215_load_format(((bc7215CombinedMsg_t*)DataToSend)->body.msg.fmt);        // 加载此命令的格式信息
                DataToSend = ((bc7215CombinedMsg_t*)DataToSend)->body.msg.datPkt;        // 将DataToSend更新为实际的数
据
                UsingSpecialFormat = 1;
            }
            bc7215_IR_tx(DataToSend);
            while (bc7215_is_busy())        // 查询是否发送完成
            {
                delay_10ms();        // 过10ms再次查询
            }

            // 指令发送完成
            if (UsingSpecialFormat == 1)
            {
                bc7215_load_format(bc7215_ac_get_base_fmt());        // 如果刚发射指令为特殊格式，则重新加载为基础格式
            }
            delay_100ms();        // 延时100ms, 确保红外指令间有足够间隔
            break;

        case AC_PARSING:                    // 解析模式
            if (bc7215_data_ready())        // 检查是否收到了数据包
            {
                IrStatus = bc7215_get_data((bc7215DataVarPkt_t*)&IrRxData[0]);        // 将所接收数据写入IrRxData[0]
                bc7215_ac_replace_base(
                    IrStatus, (bc7215DataVarPkt_t*)&IrRxData[0]);        // 用所接收数据替换基础数据包
                if (bc7215_ac_parse(&temp, &mode, &fan, &power))         // 尝试解析
                {
                    show_parse_result();        // 解析成功，显示结果
                }
                else
                {
                    show_error();        // 解析未成功，显示错误
                }
            }
            break;
        case FIND_NEXT:        // 尝试下一匹配
            if (bc7215_ac_find_next())
            {
                MatchCnt++;
```

```
                EEPROM_save_match(MatchCnt);         // 保存匹配计数，以便下次恢复
            }
            else
            {
                show_no_more_match_message();         // 显示没有更多匹配信息
                lib_init();                           // 重新初始化空调码库
            }
            break;
        case INIT:        // 重新采样初始化
            sample_and_init();
            WorkingMode = AC_CONTROL;
            break;
        default:
            break;
        }
    }
}

// 初始化空调码库
// 先尝试使用保存在EEPROM中的数据进行初始化，如果失败
// 则转去执行采样初始化
void ac_lib_init(void)
{
    IrRxData[0] = EEPROM_read_data();            // 尝试读取保存的初始化数据, 并存入IrRxData[0]
    IrRxFormat[0] = EEPROM_read_format();        // 尝试读取保存的初始化用格式信息, 并存入IrRxFormat[0]
    IrStatus = IrRxFormat[0].signature.bits.sig;        // 使用format数据中的特征字作为Status
                                                        // (因所保存的数据为从库中倒出，必为不反相的数据,
                                                        // 故不用考虑反相问题)
    IrRxMsg[0].bitLen = 0;        // 将输入存入IrRxMsg[0];
    IrRxMsg[0].body.msg.datPkt = (bc7215DataVarPkt_t*)&IrRxData[0];
    IrRxMsg[0].body.msg.fmt = &IrRxFormat[0];
    if (!bc7215_ac_init(IrStatus, (bc7215DataVarPkt_t*)&IrRxMsg[0]))        // 尝试使用保存的数据进行初始化
    {
        sample_and_init();        // 如果初始化不成功，转去执行采样和初始化
    }
    else        // 初始化成功
    {
        MatchCnt = EEPROM_read_match();        // 读取保存的匹配计数
        for (i = 0; i < MatchCnt; i++)
        {
            if (!bc7215_ac_find_next())        // 如果寻找匹配失败
            {
                show_init_match_error_message();        // 显示匹配错误
                // 这种情况下可由用户决定使用第一个匹配或重新采样初始化
                // 相应代码放在这里
            }
        }
        if (bc7215_ac_need_extra_sample() > 0)        // 检查控制协议是否需要额外格式信息,
                                                      // (保存的信息不会为多段形式(因已在初始化阶段转换),
                                                      // 故不用考虑多段格式)
        {
            IrRxData[0] = EEPROM_read_extra_data();        // 读取保存的额外数据，依然保存在IrRxData[0]中
            IrRxFormat[0] = EEPROM_read_extra_format();        // 读取保存的额外格式信息
            IrStatus = IrRxFormat[0].signature.bits.sig;
            bc7215_ac_save_2nd_base(IrStatus, &IrRxMsg[0]);        // 加载额外信息,
                                                                   // 因IrRxMsg[0]前面已经设置指向IrRxData[0]
                                                                   // 和IrRxFormat[0], 可直接使用
        }
    }
}

// 采样并初始化，此处设置为仅当初始化成功后才返回
void sample_and_init(void)
{
    uint8_t MsgCnt;
    bool    RxErr;
    while (1)        // 直到初始化成功才退出
    {
        bc7215_set_rx();              // 设置芯片为接收模式
        delay_100ms();                // 确保切换完成
        bc7215_set_rx_mode(1);        // 设置为复合模式
        show_init_message();          // 显示提示，提醒用户将遥控器设为制冷模式25度并按风力键
        MsgCnt = 0;
        RxErr = false;
        while (1)
```

```
{
    if (bc7215_format_ready())          // 检查是否收到格式包
                                        // (格式包总是在数据包之后发送，因此收到格式包必然也会收到数据包)
    {         // 如果收到格式包
        IrStatus[MsgCnt]
            = bc7215_get_data((bc7215DataVarPkt_t*)&IrRxData[MsgCnt]);         // 将收到数据和状态字存入缓存
        bc7215_get_format(&IrRxFormat[MsgCnt]);         // 收到的格式信息存入缓存
        MsgCnt++;
        if ((IrStatus[MsgCnt] & 0x80) || (MsgCnt >= 4))         // 检查接收数据是否有错误或信号段数超过限制
        {
            RxErr = true;
            break;
        }
        reset_start_timer();         // 启动计时器
    }
    else         // 如果未收到格式包
    {
        if (MsgCnt != 0)         // 如果已经收到了数据
        {
            if (bc7215_is_busy())         // 检查bc7215是否空闲
            {
                reset_start_timer();         // 如果bc7215处于忙碌状态，重新开始计时
            }
            else
            {
                if (timer > 200ms)         // 如果bc7215处于空闲且时间大于200ms
                {
                    break;
                }
            }
        }
    }
}
bc7215_set_tx();         // 退出接收模式
delay_100ms();
if (RxErr)         // 如果接收过程有错误
{
    show_error_message();         // 显示错误信息，可再次尝试或转去尝试使用预定义数据
    continue;
}
// 尝试使用收到的第一个信号进行初始化
IrRxMsg[0].bitLen = 0;         // 设置IrRxMsg[0];
IrRxMsg[0].body.msg.datPkt = (bc7215DataVarPkt_t*)&IrRxData[0];
IrRxMsg[0].body.msg.fmt = &IrRxFormat[0];
if (bc7215_ac_init(IrStatus[0], (bc7215DataVarPkt_t*)&IrRxMsg[0]))         // 尝试初始化，使用带格式的初始化，
                                                                           // 以方便后面获取基础格式包
{
    // 初始化成功
    ExtraSample = bc7215_ac_need_extra_sample();
    if (ExtraSample > 0)         // 检查是否需要额外采样
    {
        if (ExtraSample == 4)         // 此协议为多段格式，需要使用bc7215_ac_init2()进行初始化
        {
            if (MsgCnt == 1)         // 如果为多段格式但只收到了1段数据
            {
                show_error_message();         // 显示错误信息，重试或转去尝试预定义数据
                continue;
            }
            for (i = 0; i < MsgCnt; i++)         // 设置IrRxMsg
            {
                if (IrStatus[i] & 0x40)         // 检查Status中的REV位
                {
                    // 如果输出数据为反相的数据，因多段初始化bc7215_ac_init2()中不包含Status信息，
                    // 需要在使用数据前人工将反相的数据恢复
                    for (j = 0; j < (IrRxData[i].bitLen + 7) / 8; j++)
                    {
                        IrRxData[i].data[j] = ~IrRxData[i].data[j];
                    }
                }
                IrRxMsg[i].bitLen = 0;
                IrRxMsg[i].body.msg.datPkt = (bc7215DataVarPkt_t*)&IrRxData[i];
                IrRxMsg[i].body.msg.fmt = &IrRxFormat[i];
            }
            if (bc7215_ac_init2(MsgCnt, IrRxMsg, 0))
            {
```

```
            // 使用多段数据初始化成功，将数据保存，以便下次上电直接使用
            EEPROM_save_data(bc7215_ac_get_base_data());         // 多段协议初始化后会合并成一个
                                                                 // 单独数据包，可用于初始化
            EEPROM_save_format(bc7215_ac_get_base_fmt());         // 保存格式信息
            EEPROM_save_match(0);                                 // 保存匹配序号
            return;
        }
        else
        {
            show_init_err_message();        // 显示初始化失败信息
            continue;
        }
    }
    else        // 温度/模式或风力键需要进一步采样
    {
        switch (ExtraSample)
        {
        case 1:                                 // 温度
            show_sample_temp_message();         // 提示用户需要进一步采样温度键
            break;
        case 2:                                 // 模式
            show_sample_mode_message();         // 提示用户需要进一步采样模式键
            break;
        case 3:                                // 风力
            show_sample_fan_message();         // 提示需要进一步采样风力键
            break;
        default:
            break;
        }
        bc7215_set_rx();              // 设置芯片为接收模式
        delay_100ms();                // 确保切换完成
        bc7215_set_rx_mode(1);        // 设置为复合模式
        while (!bc7215_format_ready())
            ;                                                                    // 等待收到信号
        IrStatus[0] = bc7215_get_data((bc7215DataVarPkt_t*)&IrRxData[0]);        // 接收数据存入
IrRxData[0];
        bc7215_get_format(&IrRxFormat[0]);
        bc7215_set_tx();        // 退出接收模式
        delay_100ms();
        IrRxMsg[0].bitLen = 0;
        IrRxMsg[0].body.msg.datPkt = (bc7215DataVarPkt_t*)&IrRxData[0];
        IrRxMsg[0].body.msg.fmt = &IrRxFormat[0];
        if (bc7215_ac_save_2nd_base(IrStatus[0], &IrRxMsg[0]))
        {
            // 如果保存特殊格式信息成功，将其存入EEPROM供下次初始化使用
            EEPROM_save_extra_data(IrRxData[0]);
            EEPROM_save_extra_format(IrRxFormat[0]);
        }
        else
        {
            show_error_message();
            continue;
                }
            }
        }
        // bc7215_ac_need_extra_sample()返回0, 无需进一步操作，初始化完成
        // 保存初始化信息到EEPROM供下次使用
        EEPROM_save_data(bc7215_ac_get_base_data());
        EEPROM_save_format(bc7215_ac_get_base_fmt());
        EEPROM_save_match(0);
        show_init_ok_message();        // 显示初始化成功信息
        return;
    }
    else        // 尝试使用第一个数据初始化失败
    {
        // 初始化失败，显示错误信息，重新开始
        show_init_error_message();
    }
  }
}
```