

```
/*
 * 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];
bc7215DataMaxPkt_t backupBaseData; // 解析操作时备份基础数据包用
uint8_t backupStatus;
bc7215CombinedMsg_t IrRxMsg[4]; // 这个变量主要用作多段格式的采样数据初始化使用
uint8_t sampleCount; // 采样到的数据组数量

// 待发射的数据
const bc7215DataVarPkt_t* DataToSend;

/*****
* 主程序 *
* *
* 注: 本代码仅适用于摄氏度空调, 如果用于华氏度空调, *
* 需将程序内相应函数替换为其_f版本 *
*****/
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_func(set_mod_high); // 配置BC7215驱动库MOD置高函数
    bc7215_config_mod_set_low_func(set_mod_low); // 配置BC7215驱动库MOD置低函数
    bc7215_config_read_mod_func(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_func(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_func(enable_rx_int); // 配置BC7215驱动库串口中断使能函数 *
    * bc7215_config_uart_int_dis_func(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
            {
                // 因为解析操作会替换基础数据包, 故先备份下来, 以便恢复
                backupBaseData.bitLen = bc7215_ac_get_base_data()->bitLen;
                memcpy(backupBaseData.data, bc7215_ac_get_base_data()->data, (backupBaseData.bitLen + 7) / 8);
                backupStatus = bc7215_ac_get_base_fmt()->signature.bits.sig;
                ac_start_capture(); // 进入红外信号采集模式
            }
            else if (WorkingMode == AC_CONTROL) // 如果工作模式从解析模式切换为控制模式
            {
                ac_stop_capture(); // 退出信号采集模式
                bc7215_ac_replace_base(backupStatus, (bc7215DataVarPkt_t*)&backupBaseData);
                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, 表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 (check_signal_captured()) // 检查是否收到了数据包
                {
                    ac_stop_capture();
                    if (ac_parse(&Temp, &Mode, &Fan, &Power)) // 采集到的数据已经位于 IrRxData和IrRxFormat
                    {
                        show_parse_result(); // 解析成功, 显示结果
                    }
                    else
                    {
                        show_error(); // 解析未成功, 显示错误
                    }
                }
        }
    }
}

```

```

        ac_start_capture();
    }
    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
        ac_lib_init(); // 重新初始化空调码库
        bc7215_load_format(bc7215_ac_get_base_fmt()); // 加载基础格式信息到BC7215A芯片,
    }
    break;
case INIT: // 重新采样初始化
    sample_and_pairing();
    bc7215_load_format(bc7215_ac_get_base_fmt()); // 加载基础格式信息到BC7215A芯片,
    WorkingMode = AC_CONTROL;
    break;
default:
    break;
}
delay(10ms);
}
}

// 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[0] = 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[0], (bc7215DataVarPkt_t*)&IrRxMsg[0])) // 尝试使用保存的数据进行初始化
    {
        sample_and_pairing(); // 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
            }
        }
    }
}

// 开始采样
void ac_start_capture(void)
{
    sampleCount = 0;
    bc7215_set_rx();
    delay(50ms);
    bc7215_set_rx_mode(1);
    bc7215_clr_data();
    bc7215_clr_format();
}

// 停止采样
void ac_stop_capture(void)
{
    bc7215_set_tx();
    delay(50ms);
}

// 初始化(配对), 调用时应已取得采样数据
bool ac_init(void)
{
    bool acInitOK = false;
    if (sampleCount == 1)
    {
        acInitOK = bc7215_ac_init(IrStatus[0], (bc7215DataVarPkt_t*)&IrRxMsg[0]);
    }
    else if (sampleCount > 1)
    {
        for (j = 0; j < sampleCount; j++)
        {
            if (IrStatus[j] & 0x40) // 如果 "REV" 位置1, 则将每个数据字节反相
            {
                for (i = 0; i < (IrRxData[j].bitLen + 7) / 8; i++)
                {
                    IrRxData[j].data[i] = ~IrRxData[j].data[i];
                }
                IrRxData[j] &= 0xbf;
            }
        }
        acInitOK = bc7215_ac_init2(sampleCount, IrRxMsg, 0);
    }
    return acInitOK;
}

// 红外信号解析 (此函数须在成功采样后调用)
bool ac_parse(int8_t* temp, int8_t* mode, int8_t* fan, int8_t* power)
{
    int i, j;

```

```

if (sampleCount == 1)
{
    bc7215_ac_replace_base(IrStatus[0], (const bc7215DataVarPkt_t*)&IrRxData[0]);
}
else if (sampleCount > 1)
{
    for (j = 0; j < sampleCount; j++)
    {
        if (IrStatus[j] & 0x40) // if receiving status has "REV" bit set, reverse every byte of data
        {
            for (i = 0; i < (IrRxData[j].bitLen + 7) / 8; i++)
            {
                IrRxData[j].data[i] = ~IrRxData[j].data[i];
            }
            IrStatus[j] &= 0xbf;
        }
    }
    bc7215_ac_replace_base(sampleCount, (const bc7215DataVarPkt_t*)IrRxMsg);
}
return bc7215_ac_parse(temp, mode, fan, power);
}

```

// 检查是否已经完成红外信号采集

```

bool check_signal_captured(void)
{
    if (bc7215_is_busy())
    {
        // 如果BC7215A处于忙状态, 更新计时器后返回
        restart_timer();
    }
    else
    {
        if (bc7215_format_ready()) // 如果接收到数据包, 也必然收到数据包(因为格式包只在数据包后发送)
        {
            if (sampleCount < 4)
            {
                bc7215_get_format(&IrRxFormat[sampleCount]); // 读取格式包
                IrStatus[sampleCount]
                    = bc7215_get_data((bc7215DataVarPkt_t*)&IrRxData[sampleCount]); // 读取数据包

                IrRxMsg[sampleCount].bitLen = 0; // 设置复合数据指向刚收到的数据包
                IrRxMsg[sampleCount].body.msg.fmt = &IrRxFormat[sampleCount];
                IrRxMsg[sampleCount].body.msg.datPkt = (bc7215DataVarPkt_t*)&IrRxData[sampleCount];
                sampleCount++;
            }
            restart_timer();
        }
        else if (bc7215_data_ready())
        {
            // 只收到数据没收到格式, 说明没有在复合模式, 重置 Rx 模式
            bc7215_set_rx_mode(1);
            bc7215_clr_data();
            bc7215_clr_format();
            restart_timer();
        }

        if (sampleCount > 0)
        {
            // 如果已经收到了采样数据, 有200ms空闲后结束采样
            if (timer > 200ms)
            {
                return true; // 采样完成
            }
        }
    }
    return false;
}

```

// 采样并配对, 化此处设置为仅当配对成功后才返回

```

void sample_and_pairing(void)
{
    while (1) // 此处设计为直到初始化成功才退出, 实用时可考虑增加强制退出机制
    {
        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
        ac_start_capture(); // 开始采样

        // ***** 接收红外信号(信号采集) *****
        while (1) // 此处为死循环, 实用时可考虑增加强制退出机制
        {
            if (check_signal_captured()) // 等待采样结束
            {
                ac_stop_capture();

                if (ac_init()) // 采样结果已经在IrRxData和IrRxFormat
                {
                    EEPROM_save_data();
                    EEPROM_save_format();
                    EEPROM_save_match(0);
                    show_ok_message(); // 配对成功
                    break;
                }
                else // Initialization failed
                {
                    show_err_message(); // 采样配对失败, 提醒用户检查遥控器设置并重试
                }
            }
            delay(10ms);
        }
    }
}

```