目录

SmartZigbee 协议

一、协议角色的定义

（1）结点：使用该协议并实现数据交互的终端都是该协议的结点
（2）主节点：完成“组网工作”的终端称为主节点
（3）子节点：加入到一个“已存在”的网络才可以正常使用协议通讯的结点称为子节点
（4）SmartZigbee 网络，由该协议组建的 zigbee 结点互联网络

二、SmartZigbee 协议的概述

该协议是基于串口通信的类 modbus 协议，它实现了：

- 若干结点的组网。
- 主节点对子结点的管理。
- 子节点对主节点的反馈。
- 子节点对子节点的影响。
- 扩展部分。

协议的基础通讯速率为 115200kbps，该协议为智能家居典型应用而设计。

三、SmartZigbee 协议的定义

协议中分为三个主要部分，其中有，建立网络，分配网络地址，结点互联，异常处理，网络取消四个部分,下面具体定义了各个部分:

1,建立网络

建立网络，是指在一个指定应用区域内，主节点主动承担起建立 SmartZigbee 的职责，即由主节点创建一个 SmartZigbee 网络。该网络一旦创建后，子节点可立即加入到网络中。并且对网络环境进行初始化，以及创建地址表，该地址表可用来分配给每一个新加入的子节点。创建结点管理信息，表示了指定结点的各种物理信息。

2，分配网络地址

分配网络地址是指，当子节点加入到一个新的 SmartZigbee 网络中时，由子节点发出申请，然后由主节点返回一个动态的 SmartZigbee 网络地址给子节点，再由子节点发送确认信心给主节点，最后再由主节点发送一个确认信息给子节点。

3，节点互联

是指在通常情况下，子节点与主节点进行的数据通信。

4，异常处理

这部分为扩展部分，包括：

（1）子节点非正常退出网络

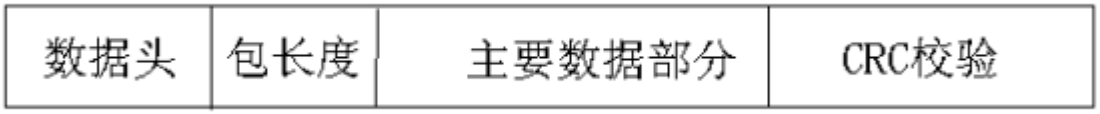
（2）主节点非正常退出网络

5，取消网络

是指主节点主动广播取消当前 SmartZigbee 网络，当子节点收到这一广播后，立刻采取 shutdown 措施，主节点同时进行相应的数据备份操作以及 shutdown 之前的工作。

四 、协议格式

SmartZigbee 协议采用了定长报文作为数据包，长度为 36 个字节（bytes）



此图为协议的总体概要图，包括 4 个部分：

(1) 数据头，它包含连个部分，一个是数据包类型，一个是子节点网络号

(2) 包长度即为有效数据的实际长度值，有效数据是指主要数据部分的值。

(3) 主要数据部分为数据包的核心数据域，当中存放的是主要传输的数据包。

(4) CRC 校验用来为使用校验算法提供参考值。

五、SmartZigbee 协议内容

FS_11C14 平台与上位机进行交互操作是依靠源文件中 datatype.h 进行的。Datatype.h 中定义了数据交互过程中所用到的所有的数据信息。该文件主要由以下几个部分构成。

（1）设备 ID 的定义

```
#define buzz_id       1          //蜂鸣器 ID 设置为 1
#define fan_id        2          //风扇 ID 设置为 2
#define seven_led_id     3          //数码管 ID 设置为 3
#define led1_id       4          //LED1 ID 设置为 4
#define led2_id       5          //LED2 ID 设置为 5
```

该部分将 FS_11C14 平台上的相关执行设备进行了设备 ID 的区分。

（2）数据包头的定义：

```
#define  start_machine_t  0xAA      //开机
#define  data_flow_t        0xBB        //数据采集
#define  rfid_msg_t       0xCC       //rfid 信息
```

```
#define  command_tag_t  0xDD       //命令
#define  key_msg_t      0xEE    //按键
#define  other_type_t       0x00       //其他（未定义）
```

在发送数据包之前，根据所要发送数据的属性在数据的前面添加一字节的包头用于标识数据信息。

（3）设备状态的定义：

```
#define led1_on      1 << 0        //标识 LED1 灯开
#define led1_off     0 << 0        //标识 LED1 灯关
#define led2_on      1 << 1        //标识 LED2 灯开
#define led2_off     1 << 0        //标识 LED2 灯关
#define fan_on       1 << 0        //标识风扇开
#define fan_off      0 << 0        //标识风扇关
#define fan_low      1 << 1        //标识风扇在低速工作
#define fan_mid      1 << 2        //标识风扇中速工作
#define fan_high     1 << 3        //标识风扇高速工作
#define speaker_on       1         //标识蜂鸣器开
#define speaker_off      0         //标识蜂鸣器关
#define seven_led_on         1          //标识数码管开
#define seven_led_off        0          //标识数码管关
```

（4）控制命令的定义：

```
#define on_led1          0         //开 LED1 灯
#define off_led1         1         //关 LED1 灯
#define on_speaker       2         //开蜂鸣器
#define off_speaker      3         //关蜂鸣器
#define on_fan           4         //开风扇
#define on_fan_low       5         //使风扇工作在低速状
态
#define on_fan_mid       6         //使风扇工作在中速状
态
#define on_fan_high      7         //使风扇工作在高速状
态
```

```
#define off_fan          8          //关风扇
#define on_seven_led     9          //开数码管
#define off_seven_led        10         //关数码管
#define off_machine      11         //关机器
```

（5）FS_11C14 平台上按键值的定义：

```
#define key_up           0          //向上方向键
#define key_down         1          //向下方向键
#define key_left             2          //向左方向键
#define key_right            3          //向右方向键
#define key_sel              4          //选择方向键
#define key_esc              5          //退出方向键
```

（6）命令类型的定义

```
#define data_r           'r'        //发送 RFID 信息
#define data_c           'c'        //接收命令信息
#define data_e           'e'        //发送环境信息
#define data_k           'k'        //发送按键信息
```

该部分定义的主要目的在于，由于数据发送的过程中，数据部分都在 data_t 结构中，定义了上面几个类型来填充不同类型的数据。

（7）各个结构体数据长度的定义：

```
#define tem_len          sizeof(tem_t)
#define hum_len          sizeof(hum_t)
#define state_len            sizeof(state_t)
#define adc_len          sizeof(adc_t)
#define acc_len          sizeof(acc_t)
#define light_len            sizeof(light_t)
#define data_len         sizeof(data_t)
#define rfid_len             sizeof(rfid_t)
#define env_len          sizeof(env_msg_t)
#define command_len      sizeof(command_t)
#define key_len          sizeof(key_t)
```

```
#define message_len      sizeof(message_t)
#define message_tag_len  sizeof(message_tag_t)
```

（8）数据结构体的定义：

```
typedef enum _message_tag_t
{
    start_machine  =  start_machine_t,     //开启设备
    data_flow      =  data_flow_t,         //数据流
    rfid_msg    =  rfid_msg_t,          //射频卡信息
    command     =  command_tag_t,       //命令
    key            =  key_msg_t,           //按键
    other_type  =  other_type_t         //其他类型
}message_tag_t;                         //标记消息类型
//该部分是所发送消息的头部，用来指明发送的是什么样的消息。
typedef struct _tem_t
{
    uint8_t ltem;
    uint8_t htem;
}tem_t;                             //温度信息

typedef struct _hum_t               //湿度信息
{
    uint8_t lhum;
    uint8_t hhum;
}hum_t;

typedef struct _light_t             //光照信息
{
    uint32_t light;
}light_t;
```

```
typedef struct _acc_t                //三轴加速度信息
{
    int8_t x;
    int8_t y;
    int8_t z;
}acc_t;

typedef struct _adc_t                //A/D 转换信息
{
    uint32_t ad0;                //data from A/D channel 0
    uint32_t ad3;            //data from A/D channel 1，
采集电池电压
}adc_t;



typedef struct _state_t          //设备状态
{
    uint8_t led_state;           //LED 灯状态
    uint8_t fan_state;           //风扇状态
    uint8_t buzz_state;          //蜂鸣器状态
    uint8_t seven_led_state;     //数码管状态
}state_t;

typedef struct _rfid_t           //RFID 相关信息，根据操作
的不同填充不同部分
{
    uint32_t id;             //序列号
    uint8_t datablock[16];       //数据块
    uint8_t purse[4];        //钱包
```

```
    uint8_t eeprom[4];       //E2PROM 值
}rfid_t;

typedef struct _command_t
{
    uint8_t operate_id;          //定义操作对象
    uint8_t operation;           //定义对象需要的操作
}command_t;



typedef struct _key_t            //M0 按键定义
{
    uint8_t key_all;
}key_t;

typedef struct _env_msg_t        //环境信息
{
    tem_t tem;                //温度
    hum_t hum;                //湿度
    acc_t acc;                    //三轴加速度
    adc_t adc;                //A/D 转换
    light_t light;                    //光照值
    state_t state;                    //设备状态
}env_msg_t;

typedef struct _other_type_d
{
    uint8_t other_data[20];          //其他数据信息
}other_type_d;
```

```
typedef union _data_t                    //可选择发送的信息类型
{
    rfid_t rfid;                    //射频卡信息
    command_t command;              //命令信息
    env_msg_t env_msg;              //环境信息
    key_t key;                  //按键信息
    other_type_d other_msg;         //其他类型信息
}data_t;

//此处定义共用体，按照数据类型不同填充不同结构向上发送。

typedef struct                          //发送消息格式
{
    message_tag_t tag;                  //消息头，区分消息类型
    uint8_t slave_address;              //从机地址，区分不同FS_11C14 设备
    uint8_t data_length;            //数据长度
    data_t data;                    //数据流
    uint16_t crc;                   //CRC 校验码
}message_t;

//最后向上层发送的消息都是以 message_t 结构向上发送的，上层接受到信息之后，按照这个协议进行解析。
```

六、流程分析

6.1 开机启动协——三次握手协议

```
int main(void)
{
```

```
    p = rx;


    init();                 //系统初始化
    fill_message(&message_s,start_machine,0,message_len,NULL,0);
    send_message_zigbee(&message_s);
/*************************************************************
    FS_11C14 上电之后，首先发送一个 message_t 结构体类型的消息，消息的头部定义为 start_machine，即第一个字节赋值为 start_machine，第二个字节即 FS_11C14 的平台地址先设置为 0. 然后通过函数 send_message_zigbee 将数据信息发送出去。
*************************************************************/
    while( 1 )
    {
/*************************************************************
下面部分为循环等待，当有接收到上层部分发送而来的消息之后，会触发 ZigBee 中断。如果没有消息到来，每隔 2 秒向上层发送开机信息。等待对方分配的节点号。
*************************************************************/
        if(zigbee_flag == 0)
        {
            memset(&message_s,message_len,0);

    fill_message(&message_s,start_machine,0,message_len,NULL,0);
            send_message_zigbee(&message_s);
            delay_ms(2000);
            continue;
        }
        else
            break;
    }


        while(1)
```

```
        {
/*************************************************************
36 字节长度的消息，即是上层发送而来的用于分配从机地址的消息。
*************************************************************/
            if(k < 36)
            {
                zigbee_flag = 0;
                ZigBee_GetChar(p);
                p++;
                k++;

            }
            else
                break;
        }
        k = 0;
        p = rx;
/*************************************************************
    判断第一个字节是不是所需要的开机消息类型，然后，如果是则将上层发送而来的消息的第二个字节取出，用来作为本机的地址。
*************************************************************/
        if(p[0] == start_machine)
        {
            STORAGE_NUM = p[1];
        }

        memset(&message_r,36,0);
        memcpy(&message_r,rx,36);
/*************************************************************
在收到消息之后，填充自己的地址信息，为确保数据的正确性，将从机地址存放在数据的第二个字节和第 4 个字节，然后将数据信息发送再一次发
```

```
送给上位机。
**********************************************************/
        memset(&message_s,message_len,0);
        p = (uint8_t *)(&message_s);
        p[3] = STORAGE_NUM;

    fill_message(&message_s,start_machine,STORAGE_NUM,message_len,
NULL,0);
        send_message_zigbee(&message_s);

        while(1)
        {
            if(zigbee_flag == 0)
                continue;
/**********************************************************
在得到主机分配的地址之后，将数据重新发给主机，由主机进行再一次的
确认，主机收到消息之后，会将信息重新发回，从机判断数据的准确性之
后，跳出握手阶段，然后进行下面的操作。
**********************************************************/
            zigbee_flag = 0;
            if((SPI752_rbuf_1[0] == start_machine) && (SPI752_rbuf_1[3]
== STORAGE_NUM))
                break;
            else
                continue;
    }
```

6.2 环境信息发送阶段：

```
if(counter1 > 2)
        {
            memset(&message_s,message_len,0);
```

```
                collect_data(&tem_s,&hum_s,&light_s);
                adc_ret(&adc_s);
                acc_ret(&acc_s);
                get_state(&state_s);
                fill_env(&env_msg_s,tem_s,hum_s,acc_s,adc_s,light_s,state_s);

                fill_data(&data_s,data_e,NULL,NULL,&env_msg_s,NULL);

    fill_message(&message_s,data_flow,STORAGE_NUM,message_len,&d
ata_s,0);
                send_message_zigbee(&message_s);
                counter1 = 0;
            }
```

首先，获取各个模块的数据信息，然后利用数据信息填充环境信息结构体 env_msg_t，然后填充数据信息结构体 data_t,最后根据是环境信息，选择正确的数据包头，填充 message_t 结构体，通过 send_message_zigbee 进行发送。利用定时器进行 counter1 数值的改变，每 2 秒触发一次操作。

6.3 接受命令发送

```
    if ((message_r.tag == command) && (message_r.slave_address ==
STORAGE_NUM))            //判断命令是否是控制本机
            {
                    switch (message_r.data.command.operate_id)
                    {
                        case on_led1:
                            GPIOSetValue(PORT3, 0, 0);          //turn on
led1
                        break;
                        case off_led1:
                            GPIOSetValue(PORT3, 0, 1);      //turn off led1
                        break;
                        case on_fan:
```

```
                    GPIOSetValue(PORT0, 2, 0);
                break;
                case off_fan:
                    GPIOSetValue(PORT0, 2, 1);
                break;
                case on_seven_led:
                    Seg7Led_Put(cnt);
                break;
                case off_seven_led:
                    Seg7Led_Put(' ');
                break;
                case on_speaker:
                    speaker_op(1);
                break;
                case off_speaker:
                    speaker_op(0);
                break;

                case off_machine://若是关机命令，则关闭所有设备，在 OLED 显示屏上打印 “I’m died” 字符。
                    GPIOSetValue(PORT3, 0, 1);
                    GPIOSetValue(PORT0, 2, 1);
                    Seg7Led_Put(' ');
                    speaker_op(0);
                    OLED_ClearScreen();
                    snprintf(dis_buf, 16, "I'm died");
                    OLED_DisStrLine(1, 0, (uint8_t *)dis_buf);
                    SysTick->CTRL                    =
~(SysTick_CTRL_CLKSOURCE_Msk |

SysTick_CTRL_TICKINT_Msk    |
```

```
SysTick_CTRL_ENABLE_Msk);//关闭系统时钟
                                for(;;){;}


                            break;


                            default:
                                printf("not write command");
                            break;


                        }
                }
/*****************************************************************
**********************
*     when the master send broadcast message,the slave will  stop working and
display     *
*     "I'm died"
                                    *
*****************************************************************
**********************/
                else if((message_r.tag == command) && (message_r.slave_address
== BROADCAST))
                {    //接收到广播命令
                    if(message_r.data.command.operate_id == off_machine)
                    {
                        GPIOSetValue(PORT3, 0, 1);
                        GPIOSetValue(PORT0, 2, 1);
                        Seg7Led_Put(' ');
                        speaker_op(0);
                        OLED_ClearScreen();
                        snprintf(dis_buf, 16, "I'm died");
                        OLED_DisStrLine(1, 0, (uint8_t *)dis_buf);
```

```
                    SysTick->CTRL                                    =
~(SysTick_CTRL_CLKSOURCE_Msk |
                                        SysTick_CTRL_TICKINT_Msk    |
                                        SysTick_CTRL_ENABLE_Msk);
                for(;;){;}
            }
        }
```

6.4 发送按键信息

```
if(up_flag)
        {
            up_flag = 0;
            fill_key(&key_s,key_up);
            fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

    fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
            send_message_zigbee(&message_s);
        }

        if(down_flag)
        {
            down_flag = 0;
            fill_key(&key_s,key_down);
            fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

    fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
            send_message_zigbee(&message_s);
        }
```

```
            if(left_flag)
            {
                left_flag = 0;
                fill_key(&key_s,key_left);
                fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

    fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
                send_message_zigbee(&message_s);
            }

            if(right_flag)
            {
                right_flag = 0;
                fill_key(&key_s,key_right);
                fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

    fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
                send_message_zigbee(&message_s);
            }

            if(sel_flag)
            {
                sel_flag = 0;
                fill_key(&key_s,key_sel);
                fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

    fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
                send_message_zigbee(&message_s);
            }
```

```
            if(esc_flag)
            {
                esc_flag = 0;
                fill_key(&key_s,key_esc);
                fill_data(&data_s,data_k,NULL,NULL,NULL,&key_s);

        fill_message(&message_s,key,STORAGE_NUM,message_len,&data_s,0)
;
                send_message_zigbee(&message_s);
            }
```

每个按键都是一个外设，在系统初始化的时候，将相应的引脚配置为中断模式，当有相关按键按下的时候，则会触发中断，在中断处理函数中，设置标志位。主程序中检测到中断标志位的改变后，会先将标志清零，然后填充相应结构体，将数据信息发送。

6.5 刷卡操作

```
if(rfid_flag)
            {
                rfid_flag = 0;
                if(Rfid_Operation(STORAGE_NUM,rbuf))
                {
                    //Speaker_OnOff(0);
                    memset(&message_s,message_len,0);
                    memcpy(&rfid_id,rbuf,sizeof(rbuf));
                    fill_rfid(&rfid_s,rfid_id,NULL,NULL,NULL);
                    fill_data(&data_s,data_r,&rfid_s,NULL,NULL,NULL);

fill_message(&message_s,rfid_msg,STORAGE_NUM,message_len,&data_s,
0);
                    send_message_zigbee(&message_s);
                }
```

```
            GPIOIntEnable(PORT2,8);

        }
```

当有射频卡操作时，会触发中断，根据标志，采集 RFID 卡的相关信息。在 Rfid_Operation 函数中，根据函数的实现不同，可以获取 RFID 的不同信息。有读卡序列号、读数据块信息、写数据块信息、读钱包值、扣钱、操作 E2PROM 等相关操作。然后利用采集到的信息填充结构体进行发送。