USER'SGUIDE

XNS102 灯控用户指南

Rev 1.0

Panchip Microelectronics

www.panchip.com

修订历史

版本	修订日期	描述
V1.0	2017-12-19	初始版本创建

目录

第 1 章 开发环境 4
1.1 软硬件开发环境 4
1.2 软件开发包 4
1.3 硬件开发包 4
1.4 开发环境搭建 4
第 2 章 方案概述 5
2.1 概述 5
2.2 硬件架构 6
第 3 章 硬件资源 9
3.1 硬件原理 9
第 4 章 软件调试 10
4.1 通信数据包格式说明 10
4.2 对码说明 10
4.3 亮度调节说明 12
4.4 色温调节说明 12
4.5 开关控制说明 13
4.6 灯状态保留功能说明 13
4.7 跳频说明 14
4.8 发射端休眠处理 15
第 5 章 软件烧录 16
5.1 烧录工具介绍 16
5.2 烧录步骤介绍 16

第 1 章 开发环境

1.1 软硬件开发环境

1) PC 端软件集成开发工具：MCU_IDE_0.80.zip(以后版本可能有更新)
2) USB 在线仿真器：应广单片机仿真 PDK5S-I-S01
3) 烧录器：应广单片机烧录器 PDK3S-P-002

1.2 软件开发包

1) XNS102_SoftwareReferenceDesign_LightControl_V1.0.zip

1.3 硬件开发包

1) XNS102_HardwareReferenceDesign_LightControl_V1.0.zip

1.4 开发环境搭建

将 MCU_IDE_0.80.zip 解压后,打开 MCU_IDE_0.80.exe,按照安装步骤提示完成安装，安装完成后桌面会出现仿真和烧录软件的图标，如下图所示。.

图 1　桌面 IDE 图标

第 2 章 方案概述

2.1 概述

本方案使用配套遥控器实现无线遥控，控制灯的开关和调节光亮，色温。具有分组控制（最高可控制四组）、灯光记忆功能（即断电后重新开启时，灯具会自动保留关机前的灯光状态）。

采用 2.4G 高频无线遥控控制，具有功耗低，传输距离远，空中通讯速率高等特点。采用跳频机制，抗干扰能力强，遥控器实物如下图所示。

图 2　遥控器按键映射图

2.2 硬件架构

1) 发射端框架图如下所示：

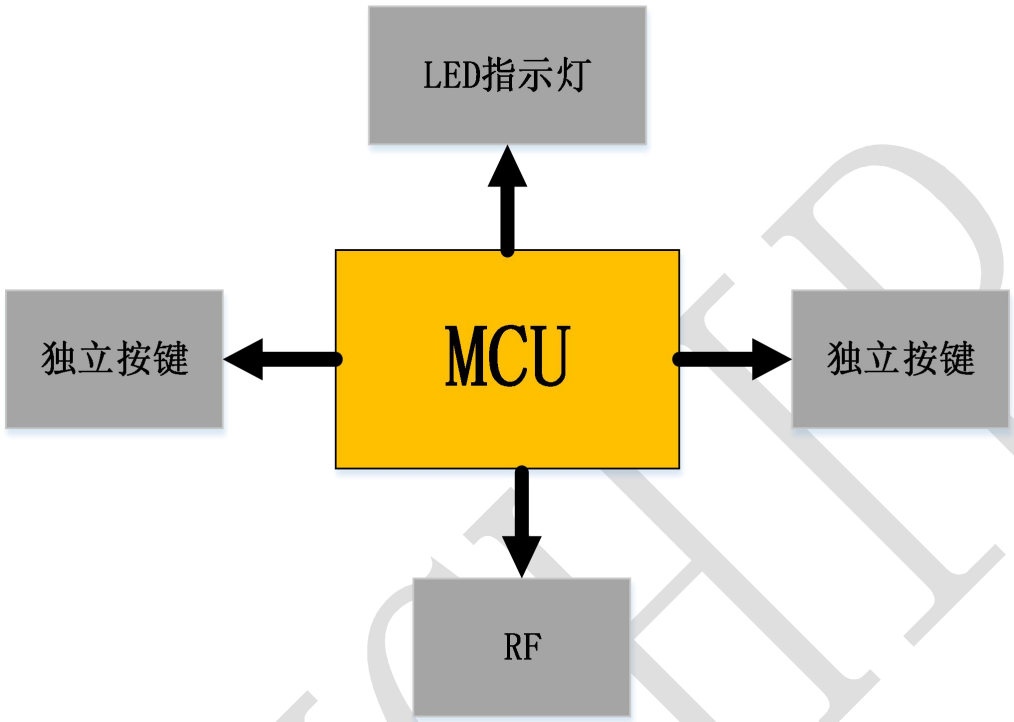


图 3　发射端框架图

2)　接收端框架图如下所示：

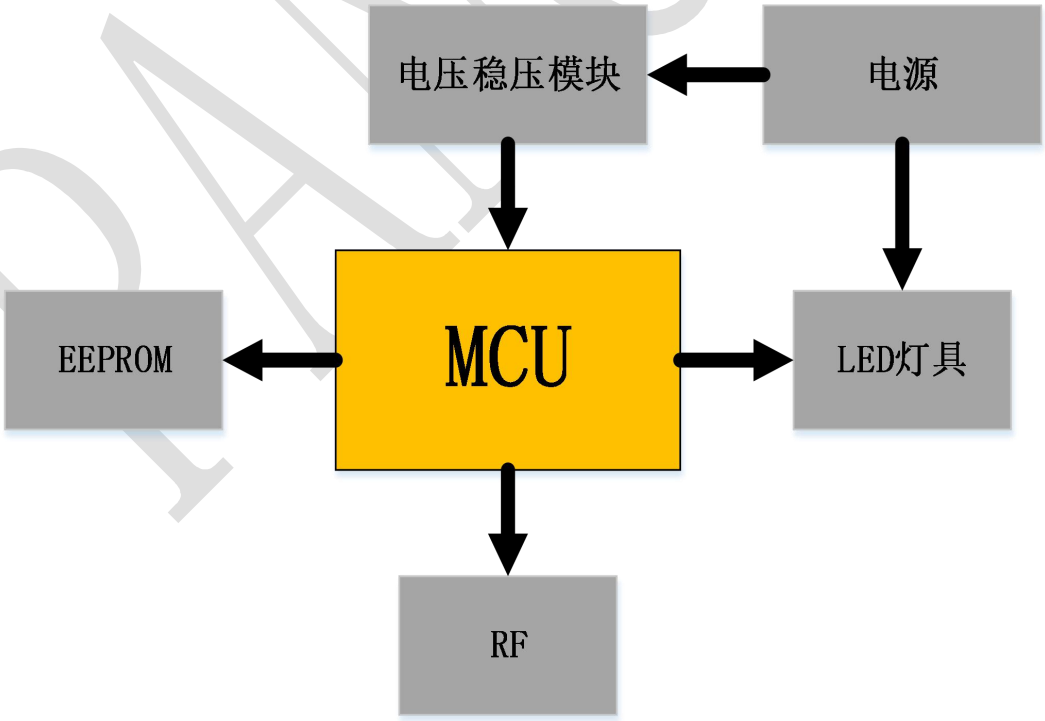


图 4　接收端框架图

2.3 软件架构

1) 发射端流程如下所示：

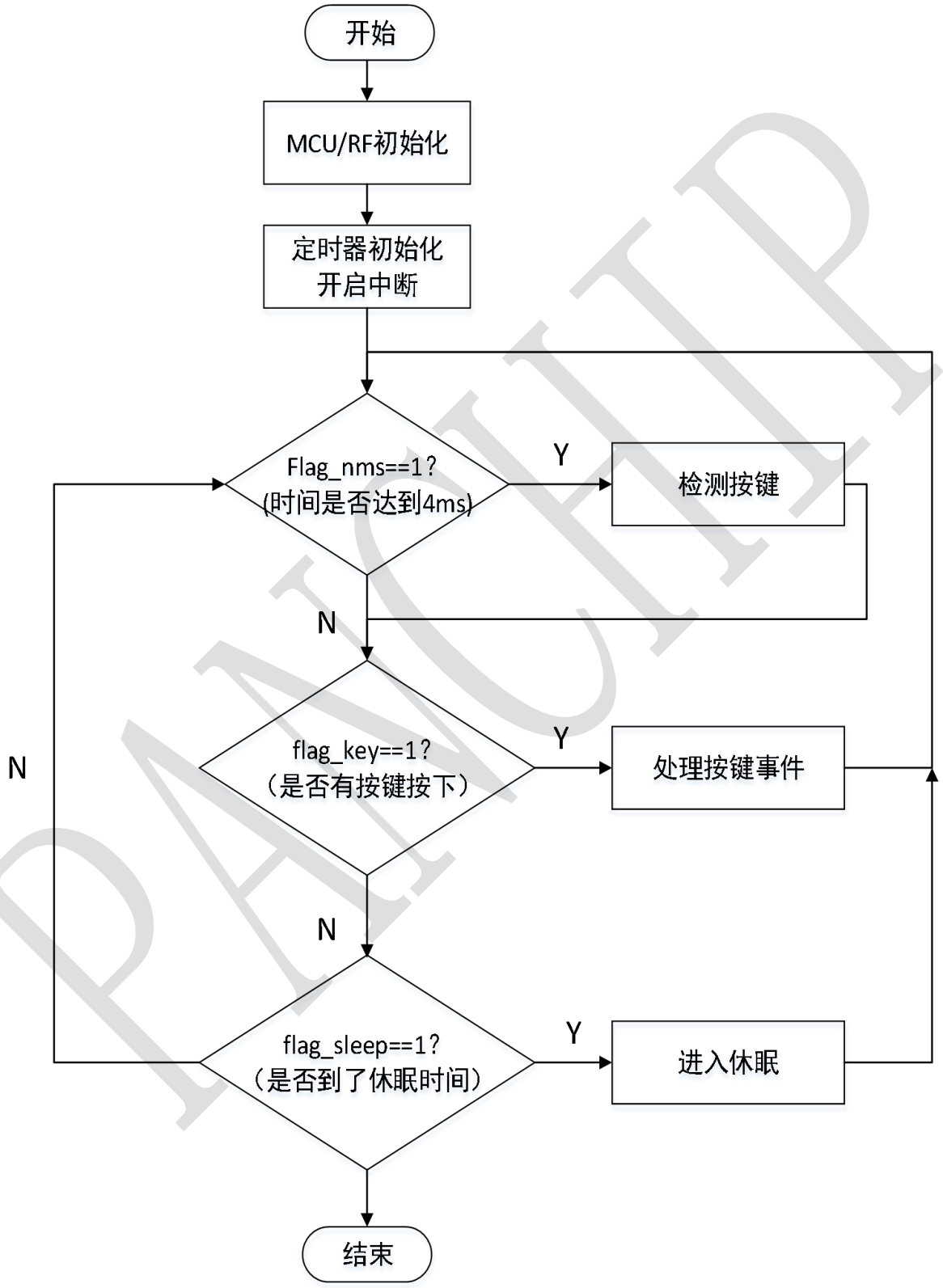


图 5 发射端流程图

2）接收端流程如下所示：

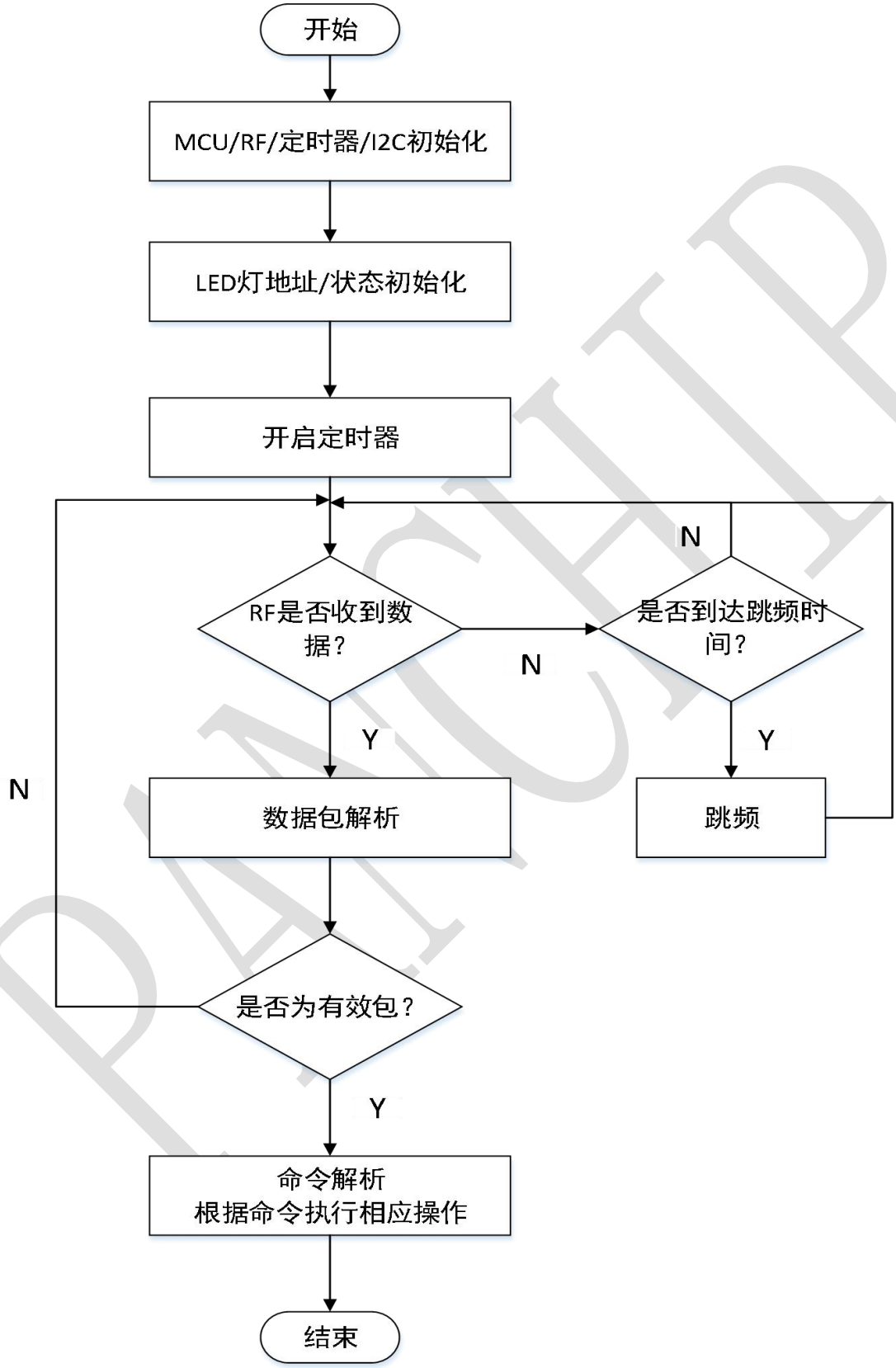


图 6　接收端流程图

第 3 章 硬件资源

3.1 硬件原理

具体请参考《硬件设计参考资料》。

第 4 章 软件调试

4.1 通信数据包格式说明

本方案中，一包数据的长度为 8bytes，数据包格式如下表所示：

表 1 通信包数据格式

字节	0	1	2	3	4	5	6	7
说明	包头			帧类型	通信频点	灯地址	功能	校验和

字节说明：

1) 包头：前三个字节为固定包头，目前用的是“PAN”。
2) 帧类型：用以区分不同的控制命令，可为如下值

0x01----------对码控制

0x03----------亮度调节

0x04----------色温调节

0x05----------开关控制

3) 通信频点：当前 RF 通信的频点。
4) 灯地址：方案中一个遥控器可以分组控制多个灯，最多四组，值可为：0x01/0x02/0x03/0x04。
5) 功能：同样的值在不同的帧类型下代表不同的操作命令，具体见后面详细介绍。
6) 校验和：整个数据包校验和。

4.2 对码说明

本方案中，发射端可以分别控制不同的四组灯（以四个不同的地址区分，地址存在接收端），也可以用一个总开关同时控制所有灯的开关（所有灯都有一个公共地址）。

实现的原理为：在 RF 通信层面上，无论发射端发送的是哪一组的控制命令，所有灯都能收到该数据包，如果判断数据包中的灯地址与自己的地址或者公共地址一致则去处理数据包，否则丢弃该包。

对码的实现操作：在接通电源的瞬间（3 秒钟内）长按相应组的开键 ，灯将"慢闪三下"确认匹配与分组成功。实现方式：

1) 发射端

```
rf_tx_buf[0]='P';
rf_tx_buf[1]='A';
rf_tx_buf[2]='N';
rf_tx_buf[3]=0x05;//帧类型
rf_tx_buf[4]=rf_channel;
rf_tx_buf[5]=node_addr;//地址
rf_tx_buf[6]=control_cmd;//数据
sum_check();
rf_tx_buf[7]=check_sum;//check_sum;
```

根据不同组的开关，发射端会将 node_addr 赋予不同的值发给接收端,接收端收到数据后将该地址设置为自己的组地址并存入 EEPROM。

2) 接收端

接收端在上电初始化的时候会读取灯之前保存的地址

```
void led_id_init(void)//LED ID初始化
{
    iic_read_bytes  IIC_DEV_ADDR,IC_CHECK_ADDR,1,&ic_check_status
    if(ic_check_status != 0x55)
    {
        ic_check_status=0x55;//写入0x55
        iic_write_bytes IIC_DEV_ADDR,IC_CHECK_ADDR,1,&ic_check_status//写入0x55
        led_dev_addr=0xff;
        iic_write_bytes IIC_DEV_ADDR,MATCH_CODE_ADDR,1,&led_dev_addr//设置ID为0xff,即为无效地址
    }
    else
    {
        iic_read_bytes  IIC_DEV_ADDR,MATCH_CODE_ADDR,1,&led_dev_addr//读取当前ID
    }
}
```

每次收到数据包后对比数据包中的地址与自身的地址是否相同

```
void rf_pkt_analysis(void)//包解析，用于分解数据包
{
    sum_check();//校验和
    if((rf_rx_buf[0] == 'P') && (rf_rx_buf[1] == 'A') && (rf_rx_buf[2] == 'N') &&
        ((rf_rx_buf[5] == led_dev_addr) || (rf_rx_buf[5] == 0x00)) && (check_sum == rf_rx_buf[PAYLOAD_WIDTH-1]))
    //帧头校,求和校验,地址校验
    {
        rf_cmd_analysis();
    }
}
```

分组地址　　公共地址

3) 清码功能

如果灯由于布局改变需要改变分组，可在灯上电的瞬间(5 秒之内),连续按相应分组开关的开键五次，灯慢闪 5 次代表清码成功，再次进行新的对码操作即可。

4.3 亮度调节说明

亮度调节的数据包格式如下：

```
rf_tx_buf[0]='P';
rf_tx_buf[1]='A';
rf_tx_buf[2]='N';
rf_tx_buf[3]=0x03;//帧类型
rf_tx_buf[4]=rf_channel;
rf_tx_buf[5]=node_addr;//地址
rf_tx_buf[6]=control_cmd;//数据
sum_check();
rf_tx_buf[7]=check_sum;
```

亮度调节的帧类型为 0x03,可控制的操作有(control_cmd)：

0xAC-------------------------亮度加(有连按)

0x5C -------------------------亮度加(无连按)

0xAA-------------------------亮度减(有连按)

0x5A -------------------------亮度减(有连按)

控制操作分为连按和不连按，在本方案中，灯的亮度调节被分为了十级(可根据不同方案作相应调整)。每个级别相对于不同的 PWM 输出数值，当不连按时，PWM 的值跳到相应的等级，有连按时，PWM 的值加一(可根据实际情况调整累加的值)。

4.4 色温调节说明

色温调节与亮度调节基本一致，对于发射端，只是帧类型的值不同。

```
rf_tx_buf[0]='P';
rf_tx_buf[1]='A';
rf_tx_buf[2]='N';
rf_tx_buf[3]=0x04;//帧类型
rf_tx_buf[4]=rf_channel;
rf_tx_buf[5]=node_addr;//地址
rf_tx_buf[6]=control_cmd;//数据
sum_check();
rf_tx_buf[7]=check_sum;
```

对于接收端，控制的命令也一样：

0xAC-------------------------亮度加(有连按)

0x5C -------------------------亮度加(无连按)

0xAA-------------------------亮度减(有连按)

0x5A -------------------------亮度减(有连按)

区别在于，如果是亮度控制，比如说亮度加，那么接收端执行的操作是冷光灯和暖光灯的 PWM 值都增加，增加到最高级别后则不作任何处理。而如果命令是色温加，执行的操作是暖光灯对于的 PWM 值增加，冷光灯不变。当暖光灯的等级到达最高级别，暖光灯值不变，冷光灯值减小。

4.5 开关控制说明

开关控制的帧类型为 0x05

```
rf_tx_buf[0]='P';
rf_tx_buf[1]='A';
rf_tx_buf[2]='N';
rf_tx_buf[3]=0x05;//帧类型
rf_tx_buf[4]=rf_channel;
rf_tx_buf[5]=node_addr;//地址
rf_tx_buf[6]=control_cmd;//数据
sum_check();
rf_tx_buf[7]=check_sum;//check_sum;
```

对于接收端，直接控制 PWM 的打开和关断来控制灯的开关

```
void led_off(void)//LED灯关闭
{
    pwmg0c=0x06;      //   PA0 OUT PWM close
    pwmg1c=0x06;      //PA4 OUT PWM close
}

void led_on(void)//LED灯开启
{
    pwmg0c=0x86;      //   PA0 OUT PWM
    pwmg1c=0x86;      //PA4 OUT PWM
}
```

4.6 灯状态保留功能说明

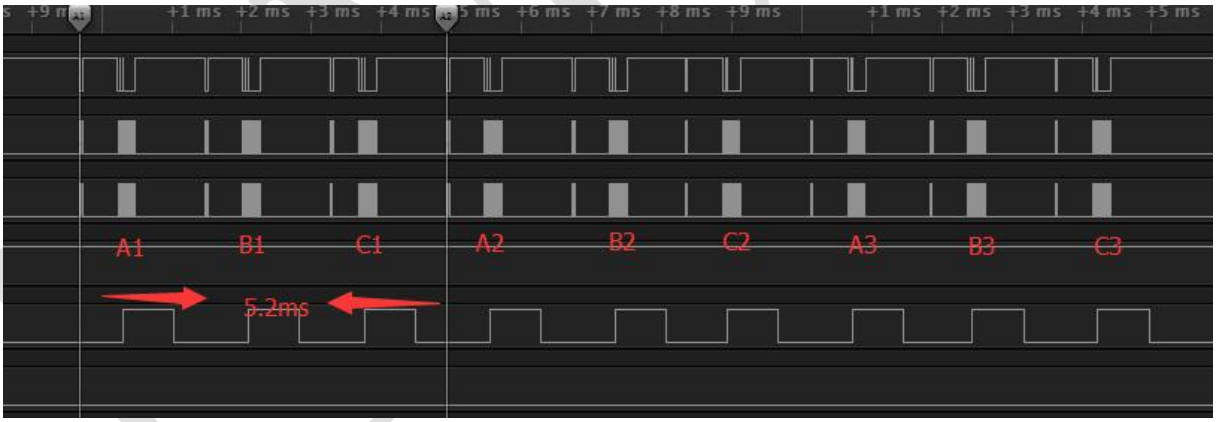
用户在把灯断电之后再次上电，灯会保留上一次断电前的状态。实现方式是在每一次对灯的亮度操作之后把对应的 PWM 值写入 EEPROM，在每次灯上电的时候将值读出来。

```
void led_status_init(void)
{
    iic_read_bytes  IIC_DEV_ADDR,LED_VALUE_CHECK_ADDR,1,&led_value_check_status
    if(led_value_check_status != 0x55)
    {
        led_value_check_status=0x55;//写入0x55
        iic_write_bytes IIC_DEV_ADDR,LED_VALUE_CHECK_ADDR,1,&led_value_check_status//写入0x55
        pwm1_duty=84;
        pwm2_duty=84;
        PWM1_INIT  0x86,256,pwm1_duty //pwmg0初始化
        PWM2_INIT  0x86,256,pwm2_duty //pwmg1初始化

        iic_write_bytes IIC_DEV_ADDR,LED1_VALUE_ADDR,1,&pwm1_duty
        iic_write_bytes IIC_DEV_ADDR,LED2_VALUE_ADDR,1,&pwm2_duty
    }
    else
    {
        iic_read_bytes  IIC_DEV_ADDR,LED1_VALUE_ADDR,1,&pwm1_duty
        iic_read_bytes  IIC_DEV_ADDR,LED2_VALUE_ADDR,1,&pwm2_duty
        PWM1_INIT  0x86,256,pwm1_duty //pwmg0初始化
        PWM2_INIT  0x86,256,pwm2_duty //pwmg1初始化
    }
}
```

4.7 跳频说明

考虑到当前的应用情况，跳频的方式采用的是不同步的方式。一共采用高中低三个频点。发射端每发送一包数据切换一次频点，按下一个键，会在每个频点发三包数据，所以一次操作一共会发 9 包数据，如下图所示：



A、B、C 代表是三个频点，1、2、3 代表的各个频点发的第几次数据。

在该应用中，接收端只要收到一包数据就可以了。事实上，九包数据全都收到跟收到一包的效果是一样的。由于不知道发射端是什么时候发数据过来，所以要确保接收端在某一个频点所处的时间要大于发射端 A、B、C 频点轮询一圈的时间（目前是 5.2ms）。程序中为了方便，目前是采用如果在 8ms 内没有接收到一包数据，则跳至下一个频点，收到数据包则重新开始计时。这种做法能确保在在干扰情况不大的情况下，能收到两包有效数据包。当然，目前的跳频

算法还是一个比较简单粗糙的算法，想要抗干扰能力更强的话，只能通过发射端发更多次数的包来保证接收端有更多的机会收到有效数据包。

4.8 发射端休眠处理

遥控器一般都是由干电池供电，所以在不工作的情况下必须进入休眠模式，以节省功耗。目前的机制是当检测按键在连续 3 秒之内没有被按下的情况下就进入休眠，如果中间有按键则重新计时。采用按键唤醒 MCU，由于该按键大部分是矩阵按键，想要确保每个按键都能唤醒系统的话，在休眠之前必须把矩阵键盘行线的电平全部拉低（矩阵键盘的列线电平都是置为高电平的，这样就满足唤醒中断触发条件）。

```
void sleep_keyboard_set(void)
{
    $   row1    out,low;
    $   row2    out,low;
    $   row3    out,low;
}
```

由于唤醒后，需要重新设置行线的 IO 口状态，也要再次初始化 RF 为 TX 模式，需要延时 10ms。这里顺序不能颠倒，必须先初始化 IO 口，再初始化 RF 模式。否则会影响按键检测的灵敏度，导致休眠后第一次按键虽然会唤醒系统，但此次的按键不会产生任何功能操作、即 RF 不会有数据发出，导致第一次的操作丢失。

第 5 章 软件烧录

5.1 烧录工具介绍

仿真器 PDK5S-I-S01，烧录器 PDK3S-P-002，选择芯片型号 XN1210B，烧录信号引脚为 PA3、PA4、PA5、PA6、MCUVDD、GND。

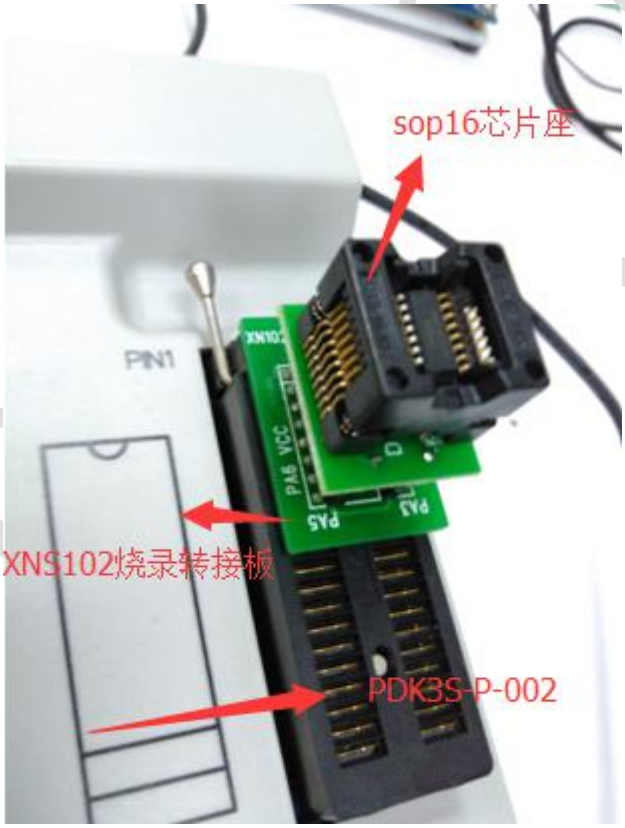
烧录过程中 MCUVDD 不能和 XN297L 的 RFVDD 相连接，建议为夹具烧录好后焊接到板子上。

5.2 烧录步骤介绍

1、硬件连接

1) 用烧录治具

首先连接好硬件，需用到的工具：PDK3S-P-002，SOP16 芯片座，XNS102 转接板，如下图示：

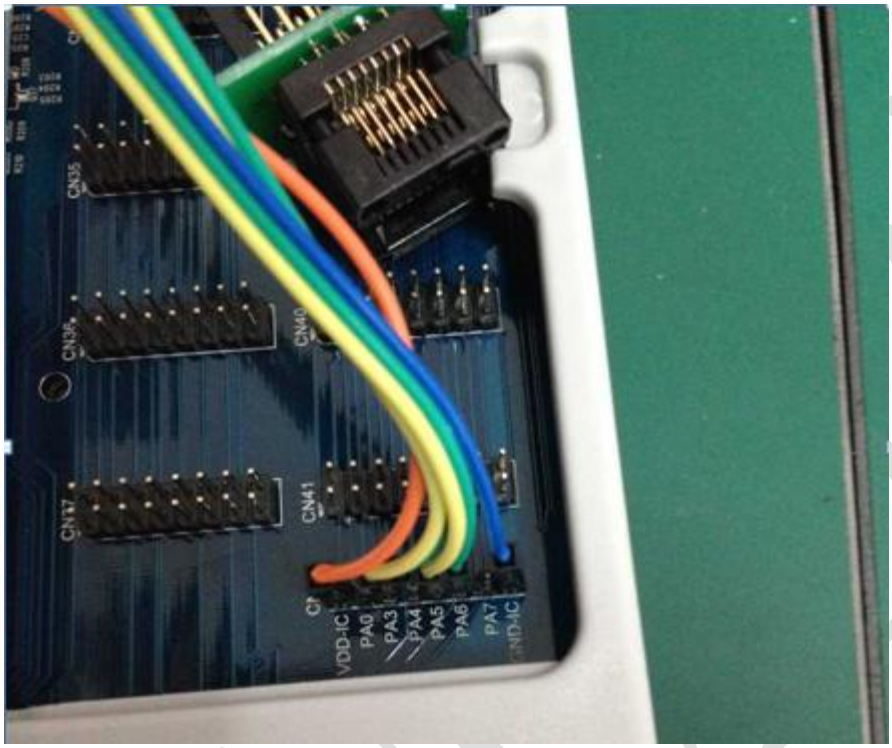


40pin—DIP 位置 1pin 为 XNS102 的 1 脚，所以转接板载烧录座上应该顶格放置。

另外，将烧录器背面的后盖打开，将跳线帽跳到 CN39 位置。

2) 连线烧录

同样将烧录器背面的后盖打开，如下图所示：



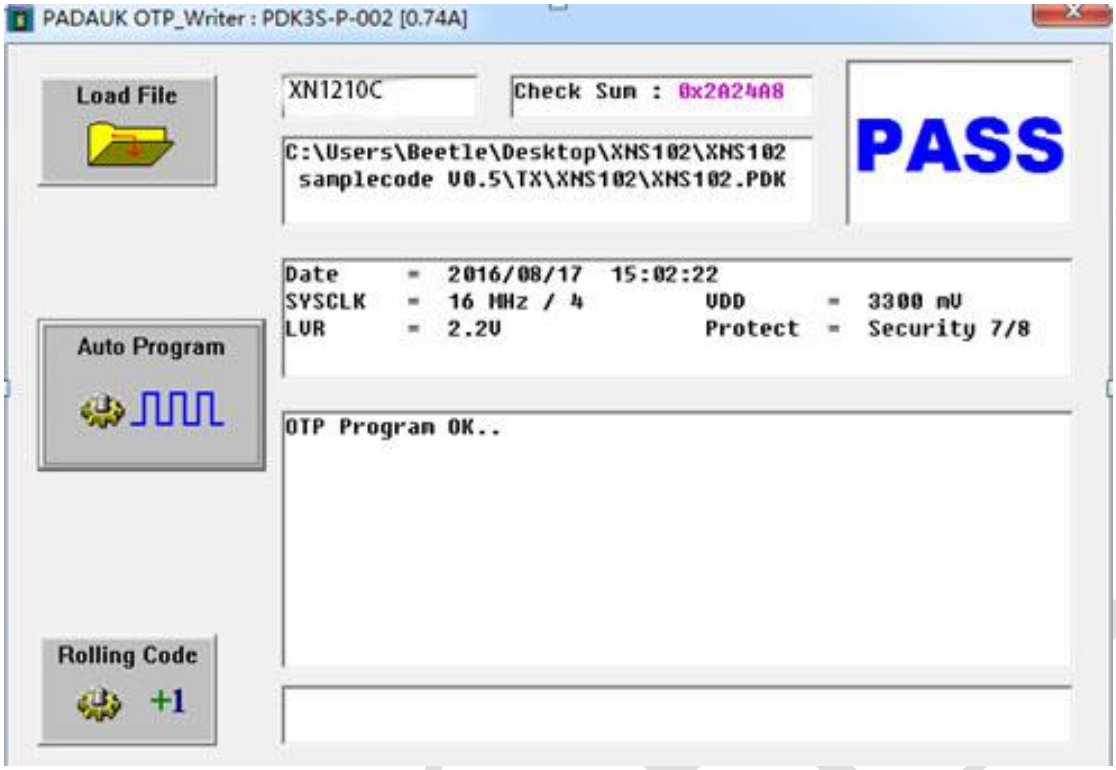
烧录器底部脚位 PA3 4 5 6 VDDIC GND 引到夹具上对应 XNS102 脚位可直接烧录，XNS102 IO 排布参考 XNS102 Datasheet 引脚顺序说明。

2、软件操作

确保硬件连接正确后，烧录器上电连接 USB，打开 

软件。

加载文件待烧录的 PDK 文档，如下图所示：



点击 Auto Program，右上角出现 PASS 图标，表示烧录成功。

如若显示 FAIL，请看软件和烧录器提示的错误信息，确认连接是否正确，芯片型号是否选择为：XN1210C。

至此烧录完成。