2.4G 无线通信使用教程

1.1 NRF24L01 无线模块简介

NRF24L01 无线模块，采用的芯片是 NRF24L01，该芯片的主要特点如下：

1）2.4G 全球开放的 ISM 频段，免许可证使用。

2）最高工作速率 2Mbps，高校的 GFSK 调制，抗干扰能力强。

3）125 个可选的频道，满足多点通信和调频通信的需要。

4）内置 CRC 检错和点对多点的通信地址控制。

5）低工作电压（1.9-3.6V）。

6）可设置自动应答，确保数据可靠传输。

该芯片通过 SPI 与外部 MCU 通信，最大的 SPI 速度可以达到 10Mhz，所以在后面软件编程的时候 SPI 速度不能高于这个最大值。本章我们用到的模块是深圳云佳科技生产的 NRF24L01，该模块已经被很多公司大量使用，成熟度和稳定性都是相当不错的。该模块的外形和引脚图如图 1.1.1 所示：

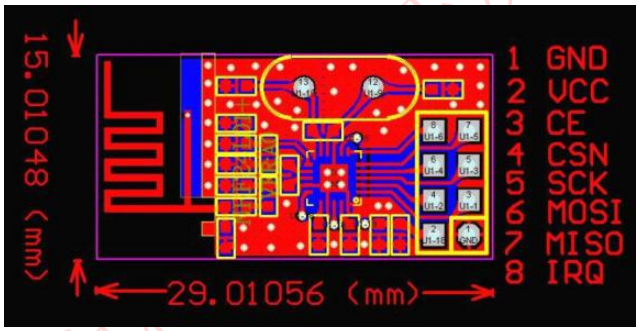

图 1.1.1 NRF24L01 模块外观引脚图

模块 VCC 脚的电压范围为 1.9-3.6V，建议不要超过 3.6V，否则可能烧坏模块，一般用 3.3V 电压比较合适。除了 VCC 和 GND 脚，其他引脚都可以和 5V 单片机的 IO 口直连，正是因为其兼容 5V 单片机的 IO，故使用上具有很大优势。关于 NRF24L01 的详细介绍，请参考 NRF24L01 的技术手册。

1.2 硬件设计

本实验功能简介：开机时系统先检测 NRF24L01 模块是否存在，在检测到 NRF24L01 模块之后，根据 K_UP 和 K_DOWN 按键来决定模块的工作模式，在设定好工作模式之后，就会开发发送/接收数据，同样用 D1 指示灯来指示程序正在运行。

开发板上并没有集成 NRF24L01 无线模块,而是预留了一个模块接口，所以我们需要知道模块接口与开发板对应的管脚原理图，如图 1.2.1 所示：

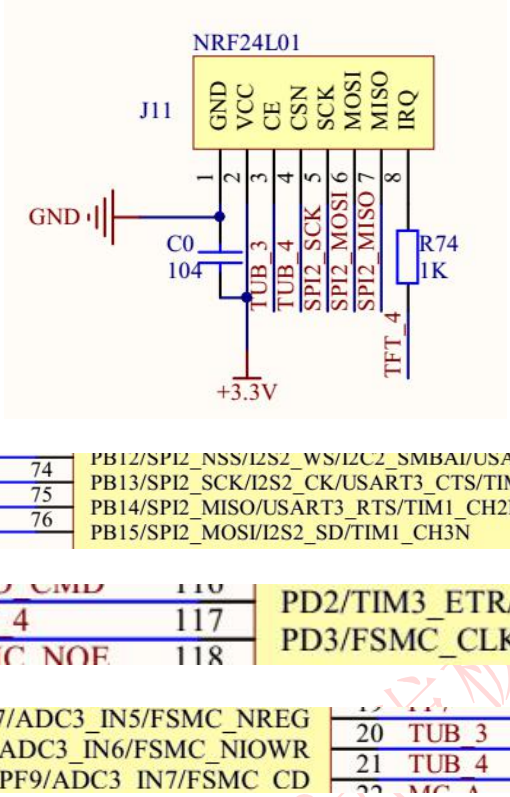

图 1.2.1 NRF24L01 模块接口与开发板连接原理图

这里 NRF24L01 模块使用的是 SPI2，和我们开发板上的 FLASH 共用一个 SPI 接口，所以在使用的时候要分时复用 SPI2。本章我们需要把 FLASH EN25QXX 的片选信号置高，以防止这个器件对 NRF24L01 的通信造成干扰。

NRF24L01 无线模块和开发板的连接实物图如图 1.2.2 所示：

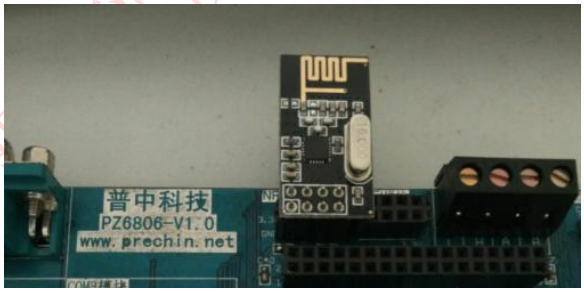

图 1.2.2 NRF24L01 模块连接图

由于 2.4G 无线通信是双向的，所以至少要有两个模块同时能工作，这里我们使用 2 套普中 STM3-PZ6806L 开发板来向大家演示。

1.3 软件设计

打开“\2.4G 无线通信应用\2.4G 无线通信程序”工程，可以看到我们加入了 nrf24l01.c 源文件和 nrf24l01.h 头文件，所有 NRF24L01 相关的驱动代码和定义都在这两个文件中实现。同时，我们还加入了之前的 spi 驱动文件 spi.c 和 spi.h 头文件，因为 NRF24L01 是通过 SPI 接口通信的。

1.3.1 NRF24L01 驱动程序

打开 nrf24l01.c 文件，代码如下：

```
#include "nrf24l01.h"
#include "spi.h"


const u8 TX_ADDRESS[TX_ADR_WIDTH]={0x34,0x43,0x10,0x10,0x01}; //发送地址
const u8 RX_ADDRESS[RX_ADR_WIDTH]={0x34,0x43,0x10,0x10,0x01};

//初始化 24L01 的 IO 口
void NRF24L01_Init(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   SPI_InitTypeDef  SPI_InitStructure;

   //使能 PB,F,D 端口时钟  //PF8-CE  PF9-CSN  PD3-IRQ
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOD, ENABLE);


   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;    //PG13 上拉 防止 EN25X 的干扰
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;          //推挽输出
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOG, &GPIO_InitStructure);//初始化指定 IO
   GPIO_SetBits(GPIOG,GPIO_Pin_13);//上拉

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;    //PB12 上拉 防止以太网 NSS 的干扰
   GPIO_Init(GPIOB, &GPIO_InitStructure);//初始化指定 IO
   GPIO_SetBits(GPIOB,GPIO_Pin_12);//上拉


   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_8; //PF8 9 推挽
   GPIO_Init(GPIOF, &GPIO_InitStructure);//初始化指定 IO
   GPIO_ResetBits(GPIOF,GPIO_Pin_9|GPIO_Pin_8);//PF6,7,8 下拉

   GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_3;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //PD3 输入
```

```
        GPIO_Init(GPIOD, &GPIO_InitStructure);
        GPIO_ResetBits(GPIOD,GPIO_Pin_3);//PD3 下拉

        SPI2_Init();          //初始化 SPI
        SPI_Cmd(SPI2, DISABLE); // SPI 外设不使能

        SPI_InitStructure.SPI_Direction                                    =
SPI_Direction_2Lines_FullDuplex;  //SPI 设置为双线双向全双工
        SPI_InitStructure.SPI_Mode = SPI_Mode_Master;     //SPI 主机
        SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;        //发送
接收 8 位帧结构
        SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;        //时钟悬空低
        SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge; //数据捕获于第 1
个时钟沿
        SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;     //NSS 信号由软件
控制
        SPI_InitStructure.SPI_BaudRatePrescaler                            =
SPI_BaudRatePrescaler_16;       //定义波特率预分频的值:波特率预分频值
为 16
        SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;  //数据传
输从 MSB 位开始
        SPI_InitStructure.SPI_CRCPolynomial = 7;  //CRC 值计算的多项式
        SPI_Init(SPI2, &SPI_InitStructure);  //根据 SPI_InitStruct 中指
定的参数初始化外设 SPIx 寄存器
        SPI_Cmd(SPI2, ENABLE); //使能 SPI 外设

        NRF24L01_CE=0;          //使能 24L01
        NRF24L01_CSN=1;         //SPI 片选取消
    }

    //检测 24L01 是否存在
    //返回值:0，成功;1，失败
    u8 NRF24L01_Check(void)
    {
        u8 buf[5]={0XA5,0XA5,0XA5,0XA5,0XA5};
        u8 i;
        SPI2_SetSpeed(SPI_BaudRatePrescaler_4); //spi 速度为 9Mhz（24L01
的最大 SPI 时钟为 10Mhz）
        NRF24L01_Write_Buf(NRF_WRITE_REG+TX_ADDR,buf,5);//写入 5 个字节
的地址.
        NRF24L01_Read_Buf(TX_ADDR,buf,5); //读出写入的地址
        for(i=0;i<5;i++)if(buf[i]!=0XA5)break;

        if(i!=5)return 1;//检测 24L01 错误
```

```
    return 0;       //检测到 24L01
}

//SPI 写寄存器
//reg:指定寄存器地址
//value:写入的值
u8 NRF24L01_Write_Reg(u8 reg,u8 value)
{
    u8 status;
        NRF24L01_CSN=0;                 //使能 SPI 传输
    status =SPI2_ReadWriteByte(reg);//发送寄存器号
    SPI2_ReadWriteByte(value);      //写入寄存器的值
    NRF24L01_CSN=1;                 //禁止 SPI 传输
    return(status);                 //返回状态值
}

//读取 SPI 寄存器值
//reg:要读的寄存器
u8 NRF24L01_Read_Reg(u8 reg)
{
    u8 reg_val;
    NRF24L01_CSN = 0;           //使能 SPI 传输
    SPI2_ReadWriteByte(reg);    //发送寄存器号
    reg_val=SPI2_ReadWriteByte(0XFF);//读取寄存器内容
    NRF24L01_CSN = 1;           //禁止 SPI 传输
    return(reg_val);            //返回状态值
}

//在指定位置读出指定长度的数据
//reg:寄存器(位置)
//*pBuf:数据指针
//len:数据长度
//返回值,此次读到的状态寄存器值
u8 NRF24L01_Read_Buf(u8 reg,u8 *pBuf,u8 len)
{
    u8 status,u8_ctr;
    NRF24L01_CSN = 0;           //使能 SPI 传输
    status=SPI2_ReadWriteByte(reg);//发送寄存器值(位置),并读取状态
值

  for(u8_ctr=0;u8_ctr<len;u8_ctr++)pBuf[u8_ctr]=SPI2_ReadWriteByte(
0XFF);//读出数据
    NRF24L01_CSN=1;         //关闭 SPI 传输
    return status;          //返回读到的状态值
```

```
    }

    //在指定位置写指定长度的数据
    //reg:寄存器(位置)
    //*pBuf:数据指针
    //len:数据长度
    //返回值,此次读到的状态寄存器值
    u8 NRF24L01_Write_Buf(u8 reg, u8 *pBuf, u8 len)
    {
       u8 status,u8_ctr;
       NRF24L01_CSN = 0;           //使能 SPI 传输
       status = SPI2_ReadWriteByte(reg);//发送寄存器值(位置),并读取状
态值
       for(u8_ctr=0; u8_ctr<len; u8_ctr++)SPI2_ReadWriteByte(*pBuf++);
//写入数据
       NRF24L01_CSN = 1;       //关闭 SPI 传输
       return status;          //返回读到的状态值
    }

    //启动 NRF24L01 发送一次数据
    //txbuf:待发送数据首地址
    //返回值:发送完成状况
    u8 NRF24L01_TxPacket(u8 *txbuf)
    {
       u8 sta;
       SPI2_SetSpeed(SPI_BaudRatePrescaler_4);//spi 速度为 9Mhz (24L01
的最大 SPI 时钟为 10Mhz)
       NRF24L01_CE=0;
       NRF24L01_Write_Buf(WR_TX_PLOAD,txbuf,TX_PLOAD_WIDTH);// 写 数 据
到 TX BUF  32 个字节
       NRF24L01_CE=1;//启动发送
       while(NRF24L01_IRQ!=0);//等待发送完成
       sta=NRF24L01_Read_Reg(STATUS);  //读取状态寄存器的值
       NRF24L01_Write_Reg(NRF_WRITE_REG+STATUS,sta); //清 除 TX_DS 或
MAX_RT 中断标志
       if(sta&MAX_TX)//达到最大重发次数
       {
          NRF24L01_Write_Reg(FLUSH_TX,0xff);//清除 TX FIFO 寄存器
          return MAX_TX;
       }
       if(sta&TX_OK)//发送完成
       {
          return TX_OK;
       }
```

```
    return 0xff;//其他原因发送失败
}

//启动 NRF24L01 发送一次数据
//txbuf:待发送数据首地址
//返回值:0，接收完成；其他，错误代码
u8 NRF24L01_RxPacket(u8 *rxbuf)
{
    u8 sta;
    SPI2_SetSpeed(SPI_BaudRatePrescaler_8); //spi 速度为 9Mhz（24L01 的最大 SPI 时钟为 10Mhz）
    sta=NRF24L01_Read_Reg(STATUS);  //读取状态寄存器的值
    NRF24L01_Write_Reg(NRF_WRITE_REG+STATUS,sta); //清除 TX_DS 或 MAX_RT 中断标志
    if(sta&RX_OK)//接收到数据
    {
        NRF24L01_Read_Buf(RD_RX_PLOAD,rxbuf,RX_PLOAD_WIDTH);//读取数据
        NRF24L01_Write_Reg(FLUSH_RX,0xff);//清除 RX FIFO 寄存器
        return 0;
    }
    return 1;//没收到任何数据
}

//该函数初始化 NRF24L01 到 RX 模式
//设置 RX 地址,写 RX 数据宽度,选择 RF 频道,波特率和 LNA HCURR
//当 CE 变高后,即进入 RX 模式,并可以接收数据了
void NRF24L01_RX_Mode(void)
{
    NRF24L01_CE=0;

  NRF24L01_Write_Buf(NRF_WRITE_REG+RX_ADDR_P0,(u8*)RX_ADDRESS,RX_ADR_WIDTH);//写 RX 节点地址

    NRF24L01_Write_Reg(NRF_WRITE_REG+EN_AA,0x01);     //使能通道 0 的自动应答
    NRF24L01_Write_Reg(NRF_WRITE_REG+EN_RXADDR,0x01);//使能通道 0 的接收地址
    NRF24L01_Write_Reg(NRF_WRITE_REG+RF_CH,40);       //设置 RF 通信频率
    NRF24L01_Write_Reg(NRF_WRITE_REG+RX_PW_P0,RX_PLOAD_WIDTH);//选择通道 0 的有效数据宽度
    NRF24L01_Write_Reg(NRF_WRITE_REG+RF_SETUP,0x0f);//设置 TX 发射参数,0db 增益,2Mbps,低噪声增益开启
```

```
    NRF24L01_Write_Reg(NRF_WRITE_REG+CONFIG, 0x0f);//配置基本工作模式的参数;PWR_UP,EN_CRC,16BIT_CRC,接收模式
    NRF24L01_CE = 1; //CE 为高,进入接收模式
}

//该函数初始化 NRF24L01 到 TX 模式
//设置 TX 地址,写 TX 数据宽度,设置 RX 自动应答的地址,填充 TX 发送数据,选择 RF 频道,波特率和 LNA HCURR
//PWR_UP,CRC 使能
//当 CE 变高后,即进入 RX 模式,并可以接收数据了
//CE 为高大于 10us,则启动发送.
void NRF24L01_TX_Mode(void)
{
    NRF24L01_CE=0;

  NRF24L01_Write_Buf(NRF_WRITE_REG+TX_ADDR,(u8*)TX_ADDRESS,TX_ADR_WIDTH);//写 TX 节点地址

  NRF24L01_Write_Buf(NRF_WRITE_REG+RX_ADDR_P0,(u8*)RX_ADDRESS,RX_ADR_WIDTH); //设置 TX 节点地址,主要为了使能 ACK

    NRF24L01_Write_Reg(NRF_WRITE_REG+EN_AA,0x01);      //使能通道 0 的自动应答
    NRF24L01_Write_Reg(NRF_WRITE_REG+EN_RXADDR,0x01); //使能通道 0 的接收地址
    NRF24L01_Write_Reg(NRF_WRITE_REG+SETUP_RETR,0x1a);//设置自动重发间隔时间:500us + 86us;最大自动重发次数:10 次
    NRF24L01_Write_Reg(NRF_WRITE_REG+RF_CH,40);       //设置 RF 通道为 40
    NRF24L01_Write_Reg(NRF_WRITE_REG+RF_SETUP,0x0f);  //设置 TX 发射参数,0db 增益,2Mbps,低噪声增益开启
    NRF24L01_Write_Reg(NRF_WRITE_REG+CONFIG,0x0e);    //配置基本工作模式的参数;PWR_UP,EN_CRC,16BIT_CRC,接收模式,开启所有中断
    NRF24L01_CE=1;//CE 为高,10us 后启动发送
}
```

此部分代码我们不多介绍，程序内有详细的注释，在这里强调一个要注意的地方，在 NRF24L01_Init 函数里面，我们调用了 SPI2_Init()函数，该函数我们在 FLASH 实验中讲到过，当时我们把 SPI 的 SCK 设置为空闲时为高，但是 NRF24L01 的 SPI 通信时序如图 1.3.1 所示：

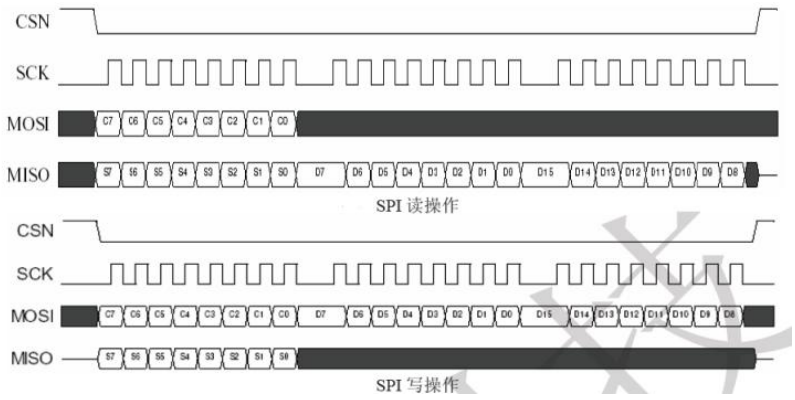

图 1.3.1 NRF24L01 SPI 通信时序图

从图中可以看出，SCK 空闲的时候是低电平的，而数据在 SCK 的上升沿被读写。所以，我们需要设置 SPI 的 CPOL 和 CPHA 均为 0，来满足 NRF24L01 对 SPI 操作的要求。所以，我们在 NRF24L01_Init 函数里面又单独添加了将 CPOL 和 CPHA 设置为 0 的代码。

接下来我们看看 nrf24l01.h 代码,该头文件主要定义了一些 NRF24L01 的命令字以及函数声明,这里还通过 TX_PLOAD_WIDTH 和 RX_PLOAD_WIDTH 决定了发射和接收的数据宽度，也就是我们每次发射和接受的有效字节数。 NRF24L01 每次最多传输 32 个字节，再多的字节传输则需要多次传送。

1.3.2 主函数

打开 main.c，代码如下：

```
/* 下载程序后，首先要按下按键 K_UP 或者 K_DOWN，按键 K_UP 是接收，K_DOWN 是发送，两块开发板
    只能一个作为发送一个作为接收，否则两个都为接收或者发送将进入死循环。接收的时候
    指示灯闪烁 NRF24L01 的最大 SPI 时钟为 10Mhz 因此在设定 SPI 时钟的时候要低于 10M*/

#include "system.h"
#include "SysTick.h"
#include "led.h"
#include "usart.h"
#include "tftlcd.h"
#include "key.h"
#include "nrf24l01.h"


void data_pros() //数据处理函数
{
   u8 key;
   static u8 mode=2;  //模式选择
   u8 rx_buf[33]="www.prechin.cn";
```

```
        static u16 t=0;
        while(1)      //等待按键按下进行选择发送还是接收
        {
            key=KEY_Scan(0);
            if(key==KEY_UP)      //接收模式
            {
                mode=0;

    LCD_ShowString(10,140,tftlcd_data.width,tftlcd_data.height,16,"RX
_Mode");

    LCD_ShowString(10,160,tftlcd_data.width,tftlcd_data.height,16,"Re
ceived Data:");

    LCD_ShowString(120,160,tftlcd_data.width,tftlcd_data.height,16,"
");
                break;
            }
            if(key==KEY_DOWN)  //发送模式
            {
                mode=1;

    LCD_ShowString(10,140,tftlcd_data.width,tftlcd_data.height,16,"TX
_Mode");

    LCD_ShowString(10,160,tftlcd_data.width,tftlcd_data.height,16,"Se
nd Data:    ");

    LCD_ShowString(120,160,tftlcd_data.width,tftlcd_data.height,16,"
");
                break;
            }
        }

        if(mode==0)       //接收模式
        {
            NRF24L01_RX_Mode();
            while(1)
            {
                if(NRF24L01_RxPacket(rx_buf)==0) //接收到数据显示
                {
                    rx_buf[32]='\0';

    LCD_ShowString(120,160,tftlcd_data.width,tftlcd_data.height,16,rx
```

```
_buf);
                    break;
                }
                else
                {
                    delay_ms(1);
                }
                t++;
                if(t==1000)
                {
                    t=0;
                    led2=~led2; //一秒钟改变一次状态
                }
            }
        }
        if(mode==1)         //发送模式
        {

            NRF24L01_TX_Mode();
            while(1)
            {
                if(NRF24L01_TxPacket(rx_buf)==TX_OK)
                {

    LCD_ShowString(120,160,tftlcd_data.width,tftlcd_data.height,16,rx
_buf);
                    break;
                }
                else
                {

    LCD_ShowString(120,160,tftlcd_data.width,tftlcd_data.height,16,"S
end Data Failed  ");

                }
            }
        }
    }

    int main()
    {
        u8 i=0;
        u16 rd=0;
        SysTick_Init(72);
```

```
        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  //中断优先级
分组 分 2 组
        LED_Init();
        USART1_Init(9600);
        TFTLCD_Init();           //LCD 初始化
        KEY_Init();
        NRF24L01_Init();


        FRONT_COLOR=BLACK;
        LCD_ShowString(10,10,tftlcd_data.width,tftlcd_data.height,16,"
PRECHIN STM32F1");
        LCD_ShowString(10,30,tftlcd_data.width,tftlcd_data.height,16,"
www.prechin.net");
        LCD_ShowString(10,50,tftlcd_data.width,tftlcd_data.height,16,"
NRF24L01 Test");
        LCD_ShowString(10,70,tftlcd_data.width,tftlcd_data.height,16,"
K_UP:RX_Mode  K_DOWN:TX_Mode");
        FRONT_COLOR=RED;

        while(NRF24L01_Check())  //检测 NRF24L01 是否存在
        {

    LCD_ShowString(140,50,tftlcd_data.width,tftlcd_data.height,16,"Er
ror   ");
        }
        LCD_ShowString(140,50,tftlcd_data.width,tftlcd_data.height,16,
"Success");

        while(1)
        {
            data_pros();
            i++;
            if(i%20==0)
            {
                led1=!led1;
            }

            delay_ms(10);

        }
    }
```

程序运行时先通过 NRF24L01_Check 函数检测 NRF24L01 是否存在，如果存在，则让用户选择发送模式(K_DOWN)还是接收模式（K_UP），在确定模式之后，

设置 NRF24L01 的工作模式，然后执行相应的数据发送/接收处理。在测试的时候一定要注意，两块开发板一个选择发送模式，一个选择接收模式，这样在 LCD 上才会显示发送的字符数据“www.prechin.cn”,还有要注意发送字节的长度，在头文件内我们已经定义了最大的发送字节长度。

1.4 实验现象

将模块连接好以后，把实验程序分别下载到两块开发板内即可看到两块开发板 LCD 显示，插上 NRF24L01 模块后，通过 K_UP 和 K_DOWN 按键，设定好对应的模式，发送端就会发送 www.prechin.com 到接收端，同时 LCD 会显示发送与接收的字符。如图 1.4.1 所示：

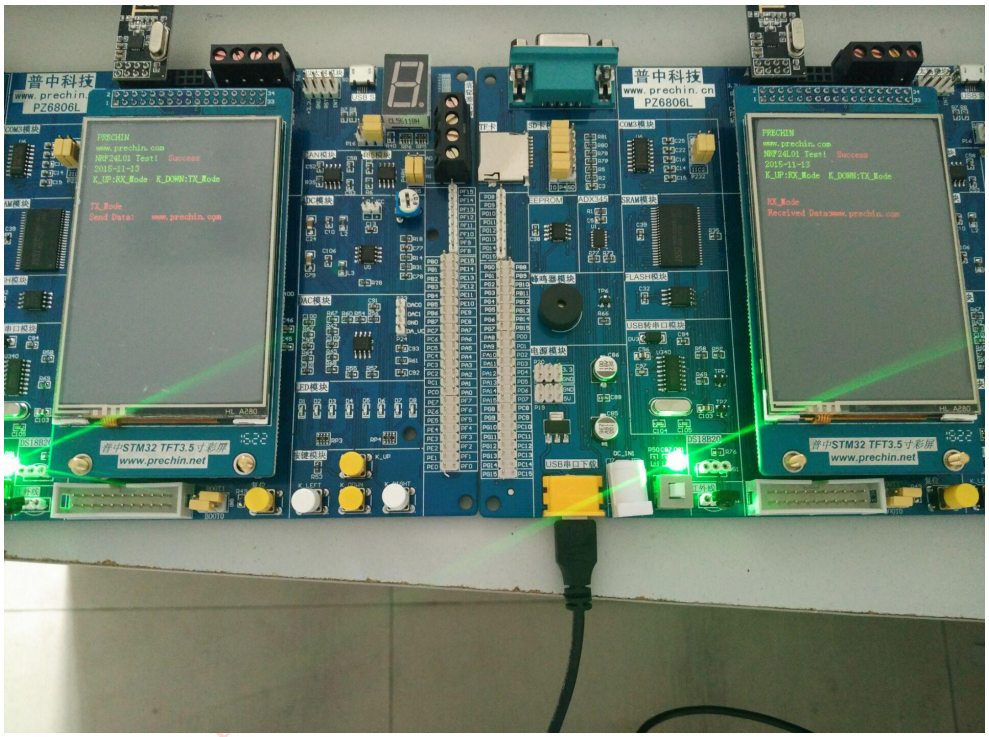

图 1.4.1 实验现象