

nRF24L01 试用实录

林新华

(江苏省苏州市, 215000)

1. 背景

因为有个潜在的项目中要使用 RFID 技术, 而且受方案成本约束, 因此考虑使用 2.4GHz 频段的芯片 nRF24L01 作为项目无线通信的 RF 核心。关于该芯片的功能就不多说了, 可以在网上找到一大堆, 选它的理由就是能满足应用要求又价格便宜。

2. 技术条件

2.1 尽量成熟的硬件平台

我在开发或者设计一样电子产品的时候, 如果要用到自己没有用过的东西, 那么初始阶段一般借用已经用过的硬件平台, 以少走弯路, 节省时间和精力, 等到自己的设计思路走通了, 再回过头来综合考虑产品的包装、开模等问题, 重新设计 PCB。因为 nRF24L01 是自己第一次使用, 所以首要的问题是找到现成的 nRF24L01 模块, 经过多方查找, 找到了一款, 该款模块使用双排针把模块的电源引脚和信号引脚全部引出, 而且 nRF24L01 的外围电路已经全部装好, 真是太好了; 第二个要借用的硬件是单片机学习板或开发板, 手头正好有几块自己做的 MSP430F135 的学习板, 上面有个排母 (14 个引脚) 原来是给 nRF905 模块用的, 现在就给 nRF24L01 模块 (8 个引脚) 对接用了, 两个模块都使用 3.3V 电源, 而且接口的电源和地正好可以对上, 唯一不足的地方就是排母的第 6 脚是悬空的, 而现在的 nRF24L01 模块却要使用该引脚! 解决的办法: 使用电烙铁把排母的第 6 脚和第 13 脚用一根短线短接。

2.2 扎实的单片机程序开发基础

其实在尝试编写读写 nRF24L01 模块的程序之前, 选择单片机平台的时候就要考虑到程序的编写问题, 使用自己最熟悉的单片机自然会使整个过程流畅不少。nRF24L01 与单片机进行的是 SPI 通信, 有两种方式可供选择: 直接使用单片机的 USCI 接口, 让其工作于 SPI 模式; 使用单片机的引脚进行模拟 SPI 通信。我在这里选择了后者。所谓“扎实的单片机程序开发基础”在这里可以找到几个观察点, 既然是扎实的, 那么肯定是做过 SPI 通信的, 既然做过 SPI 通信, 那么直接把以前的 SPI 通信的函数移植过来就可以了, 哈哈, 真的是非常节省时间。

2.3 吃透 nRF24L01 的原版 Datasheet

我的经验：芯片的中文资料有的时候会误导人！看过很多种芯片的资料，有一点是中文的，大部分是英文的，因为现在使用的芯片大多从国外进口，其说明书基本上是没有中文版本的，因此从网上找来的关于芯片的中文说明书大多是国内一些技术人员翻译的，存在理解上的误差！因此我的做法是直接登陆 NORDIC 公司的网站，找到 nRF24L01 的 Datasheet，下载，研究。先看了 1 天，大概知道了该芯片的工作模式，以及不同模式间的切换条件；然后开始编写发送程序，下载到一块学习板的 MSP430F135 中，再写接收程序，下载到另一块学习板的 MSP430F135 中，板子上电，通信，接收方没有接收到数据，失败；重新研究芯片的 Datasheet，改写程序，再试，哈哈，接收方表示接收到数据的蜂鸣器响了，成功！

3. 注意点

nRF24L01 的通信方式有 2 种：Enhanced ShockBurst™ 和 ShockBurst™，前者有自动应答机制，感觉应该适合用在无线鼠标、无线键盘等场合，而后者兼容 nRF2401A, nRF24E1, nRF2402 及 nRF24E2 的空中通信接口，而且 nRF24L01 在项目中的主要应用是有源电子标签及其读卡器，因此考虑使用 ShockBurst™ 通信方式。

nRF24L01 有 4 种工作模式：分别是 Power Down Mode, Standby Modes, RX Mode, TX Mode；控制芯片工作的信号引脚有 6 条：CE, CSN, SCK, MOSI, MISO, IRQ 具体的细节可以看芯片的 Datasheet，某些用法也可以参考我后面给出的原代码。注意的地方是：有的 nRF24L01 的命令只能用在 Power Down Mode 和 Standby Modes，不注意到这点，很容易出错；单片机和 nRF24L01 不进行 SPI 通信的时候，单片机要让 CSN 引脚为高电平 1，进行 SPI 通信的时候，nRF24L01 作为 SPI 的从设备，单片机作为 SPI 通信的主设备，通信总是从单片机控制 CSN 引脚的电位从 1 变为 0 开始，然后是单片机通过 MOSI 给出的 SPI 命令，如果还有数据则可以跟上数据，最后以 CSN 变为 1 结束本次 SPI 通信，当然如果通过 MOSI 给出的 SPI 命令是读取命令，则 nRF24L01 的应答从 MISO 引脚返回（当然每次有 SPI 命令时，nRF24L01 都会把当前自己的状态同步通过 MISO 告诉单片机）；**CE 引脚应该在什么时候置 0，什么时候置 1 很关键，自己写第一遍程序之所以失败，就是没有用好这个 CE 信号；**IRQ 信号是 nRF24L01 当满足 3 个条件中的某一个时送给单片机的提醒信号，我在自己的程序里只允许了 1 个触发条件，即接收到正确数据的时候 nRF24L01 会把 IRQ 置 0，这个 IRQ 信号需要单片机在执行完对应的操作后使用 SPI 命令让 nRF24L01 重新置 1，类似于 MSP430 单片机 P1 和 P2 口中断标志的软件清 0。

4. 程序清单

在网上搜索过 nRF24L01 的应用程序，感觉不太爽，有藏头露尾的，有要注册才给看的，有要买了他的东西才给看的！算了还是自己写吧，等调试通了，就公告天下，呵呵，会很有成就感的哦!!! NORDIC 网站上有一个文档是专门介绍如何编程应用 nRF24L01 的，硬件平台使用的单片机是 C8051F320，我没有细看就迫不及待的写程序了，现在想起来，确实没有必要看，有那个看的功夫，午饭都做好了，笑一个:)

4.1 无线数据发送的程序

4.1.1 主文件部分

文件 main.c 如下:

```
//  
//Shock Burst 通信方式, 1Mbps, 2 字节 CRC 校验, 无自动应答机制, RF 频道 2, 管道 0, 5 字节地址,  
//3 字节数据, 当有按键按下, 就发送数据出去  
#include <msp430x13x.h>  
#include "MSP430_M.h"  
#include "delay.h"  
#include "initial.h"  
#include "defineall.h"  
#include "nRF24L01.h"  
  
void Beep(unsigned int BeepCount); //蜂鸣器鸣叫函数  
unsigned char KeyScan(void); //按键扫描函数  
//  
unsigned char Tx_Payload[32]={1, 2, 3};  
unsigned char Tx_Address[5]={100, 101, 102, 103, 104};  
unsigned char ucharKeyValue=0;  
//  
int main(void)  
{  
    WDCTL = WDTW + WDTW; // Stop WDT  
    init_port(); //初始化 IO  
    init_clock(); //初始化单片机时钟: MCLK 时钟源选为 XT2CLK, 8M  
    //  
    Init_nRF24L01_IO(); //Good Beginning, Half Success.  
    nRF24L01_Setup1(); //Standby-I  
    nRF24L01_Setup2(); //  
    nRF24L01_Set_Rx_Address(nRF24L01_W_REGISTER+nRF24L01_RX_ADDR_P0, Tx_Address, 5); //No sense  
    //  
    //
```

3

```
while(1)
{
    Delay(64000); //无所谓
    ucharKeyValue=KeyScan(); //按键扫描
    if(ucharKeyValue<0xf0) //有按键输入
    {
        Beep(1); //声音提示
        nRF24L01_Flush_TX_FIFO(); //清空 FIFO
        nRF24L01_Set_Tx_Address(nRF24L01_W_REGISTER+nRF24L01_TX_ADDR, Tx_Address, 5); //指定接收
                                                    //方的地址

        Tx_Payload[0]=ucharKeyValue; //键值
        nRF24L01_Write_Tx_Payload(Tx_Payload, 3); //把3字节数据放入 FIFO
        nRF24L01_Transmit_Data(); //启动发送, 发送完成后进入 Standby-I 模式
    }
}
//
//蜂鸣器鸣叫函数
void Beep(unsigned int BeepCount)
{
    unsigned int i=0;
    for(i=0; i<BeepCount; i++)
    {
        setbit(p_beep, beep);
        Delay(60000);
        clrbit(p_beep, beep);
        Delay(60000);
    }
}
//
//按键扫描函数
unsigned char KeyScan(void)
{
    unsigned char KeyValue=0xf0;
    unsigned char Ram1=0;
    static unsigned KeyPushFlag=0;
    Ram1=(KeyInPort&0xf0);
    if(Ram1==0xf0) //没有按键输入
    {
        KeyPushFlag=0;
    }
    else //有键按着
    {
        if(KeyPushFlag==0) //上回扫描的时候没有键按着
```

4

```
{
    KeyPushFlag=1;//置标志位
    Delay(64000);//去按键抖动
    Ram1=(KeyInPort&0xf0);
    if(Ram1<0xf0)//真的有按键输入
    {
        KeyValue=Ram1;//获得键值
    }
}
return KeyValue;//返回
}
```

文件 **initial.c** 如下:

```
//
#include <msp430x13x.h>
#include "defineall.h"
/*端口初始化*/
void init_port(void)
{
    P1SEL = 0;//
    P1DIR = BIT1+BIT2+BIT3+BIT6+BIT7;
    P1OUT = BIT7;
    P2SEL = 0;//
    P2DIR = BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6;
    P2OUT = BIT0+BIT1+BIT5;
    //P2DIR = 0x00;

    P3SEL = BIT5+BIT4; // P3.4,5 = USART0 TXD/RXD
    P3DIR = BIT1+BIT2+BIT3;//
    P3OUT = BIT1;
    P4SEL = 0;
    P4DIR = BIT3;//
    P4OUT = 0;

    P5SEL = 0x00;//选择 P5 端口为 I/O 端口
    P5DIR = 0;//
    P5OUT = 0;
    P6SEL = BIT3+BIT5;
    P6DIR = BIT6+BIT7;
    P6OUT = 0;
}
//
//
```

5

```
//*****  
//时钟设置初始化  
//*****  
//在MSP430F1XX系列中,时钟模块的控制由3个寄存器来完成:DCOCTL,BCSCTL1及BCSCTL2  
void init_clock(void)  
{  
    unsigned int i;  
    BCSCTL1 =XT2OFF&(~0x80); //XT2 振荡器开启+LFXT1 低频模式+ACLK 不分频  
    do  
    {  
        clrbit(IFG1,0FIFG);  
        for (i = 0x0fff; i>0; --i);  
    }while((IFG1 & 0FIFG));  
  
    BCSCTL2 = SELM_2+SELS;//MCLK 时钟源选为 XT2CLK, 8M  
  
}  
//
```

文件 **delay.c** 如下:

```
//  
void Delay(unsigned int time)  
{  
    unsigned int i;  
    for(i=time;i>0;i--)  
    {}  
}  
//
```

文件 **nRF24L01.c** 如下:

```
//  
#include "defineall.h"  
#include "delay.h"  
#include <msp430x13x.h>  
//  
void Init_nRF24L01_IO(void)//  
{  
    nRF24L01_CE_OutDIR();  
    nRF24L01_CSN_OutDIR();  
    nRF24L01_SCK_OutDIR();  
    nRF24L01_MOSI_OutDIR();  
    nRF24L01_MISO_InDIR();  
    nRF24L01_IRQ_InDIR();  
}  
//
```

6

```
nRF24L01_Clear_CE();
nRF24L01_Set_CSN();
nRF24L01_Clear_SCK();
nRF24L01_Clear_MOSI();
}
//
//
//function SpiWrite();
/*****/
void nRF24L01SpiWrite(unsigned char byte)
{
    unsigned char i;
    nRF24L01_Clear_SCK();
    Delay(1);
    for (i=0;i<8;i++) // Setup byte circulation bits
    {
        if ((byte&BIT7)==BIT7) // Put DATA_BUF.7 on data line
            nRF24L01_Set_MOSI(); //MOSI=1;
        else
            nRF24L01_Clear_MOSI(); //MOSI=0;
        nRF24L01_Set_SCK(); // Set clock line high
        Delay(2);
        byte=byte<<1;
        nRF24L01_Clear_SCK(); // Set clock line low
        Delay(2);
    }
    Delay(1);
}
//
//
//function SpiRead();
/*****/
unsigned char nRF24L01SpiRead(void)
{
    unsigned char i;
    unsigned char temp=0;
    nRF24L01_Clear_SCK();
    Delay(2);
    for (i=0;i<8;i++) // Setup byte circulation bits
    {
        nRF24L01_Set_SCK(); // Set clock line high
        Delay(2);
        temp=temp<<1; // Right shift DATA_BUF
        if ((nRF24L01_MISO_InPort&nRF24L01_MISO)==nRF24L01_MISO)
```

```
        {temp|=1;}           // Read data
        nRF24L01_Clear_SCK(); // Set clock line low
        Delay(2);
    }
    Delay(2);
    return temp;             // Return function parameter
}
//
//
void nRF24L01_Flush_TX_FIFO(void)//Clear TX FIFO
{
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(nRF24L01_FLUSH_TX);
    nRF24L01_Set_CSN();
}
//
//
void nRF24L01_Flush_RX_FIFO(void)//Clear RX FIFO
{
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(nRF24L01_FLUSH_RX);
    nRF24L01_Set_CSN();
}
//
//
void nRF24L01SpiWriteReg(unsigned char SpiCommand,unsigned char Content)
{
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(SpiCommand);
    nRF24L01SpiWrite(Content);
    nRF24L01_Set_CSN();
}
//
//
void nRF24L01_Set_Rx_Address(unsigned char RX_Address_Pipex,unsigned char *Address,unsigned char Length)//Local
                                                                    //Address
{
    unsigned char i=0;
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(RX_Address_Pipex);
```

8


```
for(i=0;i<Length;i++)
{
    nRF24L01SpiWrite(*Address);
    Address++;
}
nRF24L01_Set_CSN();
}
//
//
void nRF24L01_Set_Tx_Address(unsigned char TX_AddressReg10,unsigned char *Address,unsigned char Length)//Remote
//Address
{
    unsigned char i=0;
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(TX_AddressReg10);
    for(i=0;i<Length;i++)
    {
        nRF24L01SpiWrite(*Address);
        Address++;
    }
    nRF24L01_Set_CSN();
}
//
//
void nRF24L01_Read_Rx_Payload(unsigned char *DataBuff,unsigned char Length)//Payload Out
{
    unsigned char i=0;
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(nRF24L01_R_RX_PAYLOAD);
    for(i=0;i<Length;i++)
    {
        *DataBuff=nRF24L01SpiRead();
        DataBuff++;
    }
    nRF24L01_Set_CSN();
}
//
//
//
//
//
//
//
```

9

```
void nRF24L01_Write_Tx_Payload(unsigned char *DataByte, unsigned char Length)//Payload IN
{
    unsigned char i=0;
    nRF24L01_Set_CSN();
    nRF24L01_Clear_CSN();
    nRF24L01SpiWrite(nRF24L01_W_TX_PAYLOAD);
    for(i=0;i<Length;i++)
    {
        nRF24L01SpiWrite(*DataByte);
        DataByte++;
    }
    nRF24L01_Set_CSN();
}
//
//
void nRF24L01_Transmit_Data(void)//Transmit Pulse
{
    Delay(8000);
    nRF24L01_Set_CE();
    Delay(8000);
    nRF24L01_Clear_CE();
}
//
//
void nRF24L01_Reset_Tx_DS(void)
{
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_STATUS, 0x2e);///
}
//
//
void nRF24L01_Reset_Rx_DS(void)
{
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_STATUS, 0x4e);///
}
//
//
void nRF24L01_Setup1(void)//Setup1
{
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_CONFIG, 0x3e);//Reflect RX_DR
                                //Mask interrupt caused by TX_DS;
                                //Mask interrupt caused by MAX_RT;
                                //Enable CRC;
                                //CRC encoding scheme 2 bytes;
                                //POWER UP;
```

10

```

//PTX;

Delay(16000); //给点时间
}
//
void nRF24L01_Setup2(void) //Setup2
{
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_EN_AA, 0x00); //Disable auto_acknowledgment, 6 pipes
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_EN_RXADDR, 0x01); //Enabled RX Addresses. Enable data pipe 0
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_SETUP_AW, 0x03); //RX/TX Address field width 5 bytes.
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_SETUP_RETR, 0x00); //Re-Transmit disabled.
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RF_CH, 0x02); //RF Channel.
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RF_SETUP, 0x06); //Air Data Rate 1Mbps. RF_PWR:0dBm. LNA Gain 0.

    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P0, 0x03); //Pipe0 3 Bytes Payload.
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P1, 0x00); //Pipe1
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P2, 0x00); //Pipe2
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P3, 0x00); //Pipe3
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P4, 0x00); //Pipe4
    nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_RX_PW_P5, 0x00); //Pipe5
}

```

4. 1. 2 头文件部分

11

头文件 `defineall.h` 如下:

```

//
#include "MSP430_M.h"
//
//蜂鸣器
#define beep BIT3
#define p_beep P4OUT
//
//按键
#define KeyInPort P4IN
//
//nRF24L01 Definitions
#define nRF24L01_CE BIT4//P2. 4
#define nRF24L01_CSN BIT5//P2. 5
#define nRF24L01_SCK BIT6//P2. 6
#define nRF24L01_MOSI BIT3//P3. 3
#define nRF24L01_MISO BIT7//P3. 7
#define nRF24L01_IRQ BIT6//P3. 6
//
//

```

```
#define NRF24L01_CE_DIR      P2DIR
#define NRF24L01_CSN_DIR    P2DIR
#define NRF24L01_SCK_DIR    P2DIR
#define NRF24L01_MOSI_DIR   P3DIR
#define NRF24L01_MISO_DIR   P3DIR
#define NRF24L01_IRQ_DIR    P3DIR
//
#define NRF24L01_CE_OutPort  P2OUT
#define NRF24L01_CSN_OutPort P2OUT
#define NRF24L01_SCK_OutPort P2OUT
#define NRF24L01_MOSI_OutPort P3OUT
#define NRF24L01_MISO_InPort  P3IN
#define NRF24L01_IRQ_InPort   P3IN
//
#define NRF24L01_CE_OutDIR()  setbit(NRF24L01_CE_DIR, NRF24L01_CE)
#define NRF24L01_CSN_OutDIR() setbit(NRF24L01_CSN_DIR, NRF24L01_CSN)
#define NRF24L01_SCK_OutDIR() setbit(NRF24L01_SCK_DIR, NRF24L01_SCK)
#define NRF24L01_MOSI_OutDIR() setbit(NRF24L01_MOSI_DIR, NRF24L01_MOSI)
#define NRF24L01_MISO_InDIR() clrbit(NRF24L01_MISO_DIR, NRF24L01_MISO)
#define NRF24L01_IRQ_InDIR()  clrbit(NRF24L01_IRQ_DIR, NRF24L01_IRQ)
//
#define NRF24L01_Set_CE()     setbit(NRF24L01_CE_OutPort, NRF24L01_CE)
#define NRF24L01_Clear_CE()   clrbit(NRF24L01_CE_OutPort, NRF24L01_CE)
#define NRF24L01_Set_CSN()    setbit(NRF24L01_CSN_OutPort, NRF24L01_CSN)
#define NRF24L01_Clear_CSN()  clrbit(NRF24L01_CSN_OutPort, NRF24L01_CSN)
#define NRF24L01_Set_SCK()    setbit(NRF24L01_SCK_OutPort, NRF24L01_SCK)
#define NRF24L01_Clear_SCK()  clrbit(NRF24L01_SCK_OutPort, NRF24L01_SCK)
#define NRF24L01_Set_MOSI()   setbit(NRF24L01_MOSI_OutPort, NRF24L01_MOSI)
#define NRF24L01_Clear_MOSI() clrbit(NRF24L01_MOSI_OutPort, NRF24L01_MOSI)
//
//SPI Commands
#define NRF24L01_R_REGISTER   0x00
#define NRF24L01_W_REGISTER   0x20
#define NRF24L01_R_RX_PAYLOAD 0x61
#define NRF24L01_W_TX_PAYLOAD 0xA0
#define NRF24L01_FLUSH_TX     0xE1
#define NRF24L01_FLUSH_RX     0xE2
#define NRF24L01_REUSE_TX_PL  0xE3
#define NRF24L01_NOP          0xFF
//
//
//
//
//
```

12

```

//Register Definitions
#define NRF24L01_CONFIG          0x00
#define NRF24L01_EN_AA          0x01
#define NRF24L01_EN_RXADDR      0x02
#define NRF24L01_SETUP_AW       0x03
#define NRF24L01_SETUP_RETR     0x04
#define NRF24L01_RF_CH          0x05
#define NRF24L01_RF_SETUP       0x06
#define NRF24L01_STATUS         0x07
#define NRF24L01_OBSERVE_TX     0x08
#define NRF24L01_CD              0x09
#define NRF24L01_RX_ADDR_P0     0x0A
#define NRF24L01_RX_ADDR_P1     0x0B
#define NRF24L01_RX_ADDR_P2     0x0C
#define NRF24L01_RX_ADDR_P3     0x0D
#define NRF24L01_RX_ADDR_P4     0x0E
#define NRF24L01_RX_ADDR_P5     0x0F
#define NRF24L01_TX_ADDR        0x10
#define NRF24L01_RX_PW_P0       0x11
#define NRF24L01_RX_PW_P1       0x12
#define NRF24L01_RX_PW_P2       0x13
#define NRF24L01_RX_PW_P3       0x14
#define NRF24L01_RX_PW_P4       0x15
#define NRF24L01_RX_PW_P5       0x16
#define NRF24L01_FIFO_STATUS     0x17

```

13

头文件 **MSP430_M.h** 如下:

```

//
//*****
#include <msp430x13x.h>
//*****
/*****/

#define clrbit(reg,bit)  reg &= ~(bit)  /*清寄存器的某1比特位*/
#define bitclr(reg,bit)  reg &= ~(bit)  /*清寄存器的某1比特位*/

#define setbit(reg,bit)  reg |= (bit)   /*设置寄存器的某1比特位*/
#define bitset(reg,bit)  reg |= (bit)   /*设置寄存器的某1比特位*/
#define cplbit(reg,bit)  reg ^= (bit)   /*对寄存器的某1比特位取反*/
#define bitcpl(reg,bit)  reg ^= (bit)   /*对寄存器的某1比特位取反*/

```

头文件 **delay.h** 如下:

```

//
extern void Delay(unsigned int time);

```

头文件 **initial.h** 如下:

```
//
extern void init_clock(void);
extern void init_port(void);
```

头文件 **nRF24L01.h** 如下:

```
//
extern void Init_nRF24L01_IO(void);
extern void nRF24L01SpiWrite(unsigned char byte);
extern unsigned char nRF24L01SpiRead(void);
extern void nRF24L01_Setup1(void);
extern void nRF24L01_Setup2(void);
extern void nRF24L01_Set_Rx_Address(unsigned char RX_Address_Pipex, unsigned char *Address, unsigned char Length);
extern void nRF24L01_Read_Rx_Payload(unsigned char *DataBuff, unsigned char Length);
extern void nRF24L01_Set_Tx_Address(unsigned char TX_AddressReg10, unsigned char *Address, unsigned char Length); //
extern void nRF24L01_Write_Tx_Payload(unsigned char *DataByte, unsigned char Length);
extern void nRF24L01_Transmit_Data(void);
extern void nRF24L01_Reset_Tx_DS(void);
extern void nRF24L01_Reset_Rx_DS(void);
extern void nRF24L01_Flush_TX_FIFO(void);
extern void nRF24L01_Flush_RX_FIFO(void);
extern void nRF24L01SpiWriteReg(unsigned char SpiCommand, unsigned char Content);
```

14

4.2 无线数据接收的程序

与前面的数据发送部分程序相比,接收部分的程序仅有少量的不同:main.c 文件中的程序不同;nRF24L01.c 文件的函数 void nRF24L01_Setup1(void) 内容不同。其他部分的文件结构和内容是一样的,因此在此就把不同的部分给出即可。

文件 **main.c** 如下:

```
//
//Shock Burst 通信方式, 1Mbps, 2 字节 CRC 校验, 无自动应答机制, RF 频道 2, 管道 0, 5 字节地址,
//3 字节数据, 当查询到有无线数据进来的时候, 读出

#include <msp430x13x.h>
#include "MSP430_M.h"
#include "delay.h"
#include "initial.h"
#include "defineall.h"
#include "nRF24L01.h"

void Beep(unsigned int BeepCount);
//
//
//
```

```
unsigned char Rx_Address[5]={100, 101, 102, 103, 104}; //本机地址
unsigned char Rx_Buff[32];
int main(void)
{
    WDCTL = WDTW + WDTW; // Stop WDT
    init_port();
    init_clock();
    Init_nRF24L01_I0();
    nRF24L01_Setup2();//
    nRF24L01_Set_Rx_Address(nRF24L01_W_REGISTER+nRF24L01_RX_ADDR_P0, Rx_Address, 5); //
    nRF24L01_Setup1();//
    nRF24L01_Flush_RX_FIFO();
    nRF24L01_Set_CE();//进入 RX 模式
    Beep(2);
    while(1)
    {
        if((nRF24L01_IRQ_InPort&nRF24L01_IRQ)!=nRF24L01_IRQ)//如果有无线数据
        {
            nRF24L01_Reset_Rx_DS();//清标志位
            nRF24L01_Read_Rx_Payload(Rx_Buff, 3);//读取数据，数据内容可以在 Debug 状态小查看
            nRF24L01_Flush_RX_FIFO();
            Delay(64000);
            Beep(1);//提示音
        }
        Delay(64000);
    }
}
//
//
void Beep(unsigned int BeepCount)
{
    unsigned int i=0;
    for(i=0;i<BeepCount;i++)
    {
        setbit(p_beep, beep);
        Delay(60000);
        clrbit(p_beep, beep);
        Delay(60000);
    }
}
```

15

文件 **nRF24L01.c** 中的 void nRF24L01_Setup1(void) 如下:

```
//
void nRF24L01_Setup1(void)//
```

```
{
nRF24L01SpiWriteReg(nRF24L01_W_REGISTER+nRF24L01_CONFIG, 0x3f); //Reflect RX_DR;
//Mask interrupt caused by TX_DS;
//Mask interrupt caused by MAX_RT;
//Enable CRC;
//CRC encoding scheme 2 bytes;
//POWER UP;
//PRX;

Delay(16000);
}
```

5. 总结与展望

试用 nRF24L01，初战告捷，得到的经验有 3 条。1，使用自己熟悉的硬件平台，使用经过验证合格的硬件模块，不在硬件设计上浪费时间；2，使用 C 语言编程，养成良好的编程习惯，注意积累，便于以后程序的移植；3，弄懂弄通 nRF24L01 的 Datasheet 上的内容，有条件的最好看英文原版。

在准备好硬件之后，从自己细看 Datasheet 到初步调试成功，花了有 5 天时间，有点仓促，程序的功能不完备还有待扩充，在以后的项目开发中还要继续优化部分函数。

6. 撰写本文的目的

16

我撰写这篇文章的目的可以归纳为 2 个：1，给自己做的 MSP430F135 单片机学习板做个广告，给自己的兼职工作打个宣传，期望可以接到更多的项目；2，给正在学习 nRF24L01 的工程师提供个程序参考，希望有所帮助。

7. 实物照片



图 1 无线通信模块 nRF24L01

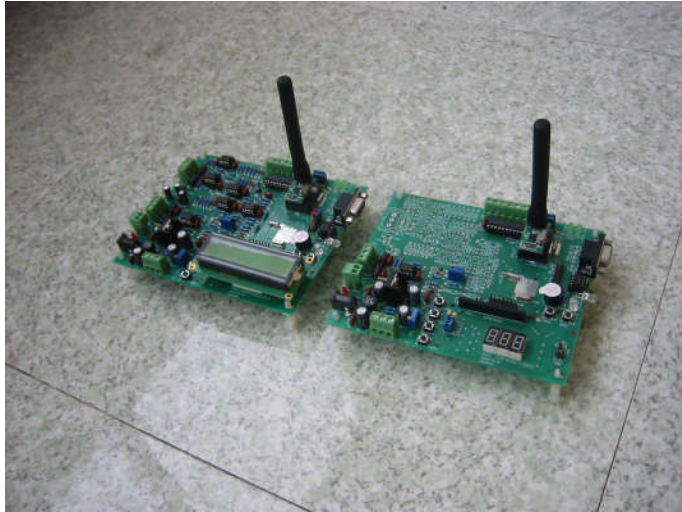


图2 MSP430F135 硬件平台

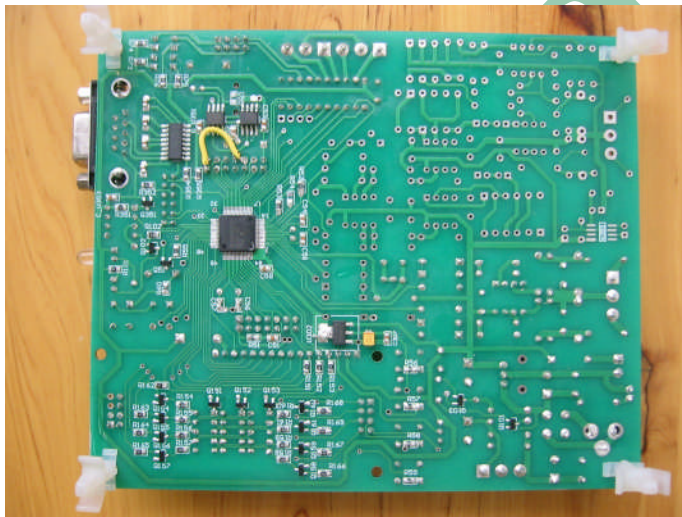


图3 硬件平台的背面（注意有根短接线）

更多照片请关注我的 QQ 空间中的相册：<http://279361937.qzone.qq.com/>

需要购买学习板的朋友可以联系我，支持支付宝和银行汇款；

当前可以承接的项目方向有：大功率开关电源设计方向；有源射频识别（RFID）应用方面（433MHz, 2.4GHz）；GSM 手机短信模块的应用；相关电脑软件如有需要也可以开发。