青风手把手教你学 stm32f051 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.21 无线 RF24L01 的读写

2.21.1 原理分析

Nrf24L01 是具有 2.4GHZ 的单片高速 2Mbps 无线收发芯片，是在无线领域使用比较广泛的无线设计方案。因此，我们在青风 stm32f0 开发板上加入了无线传输模块接口，用于大家做无线实验。

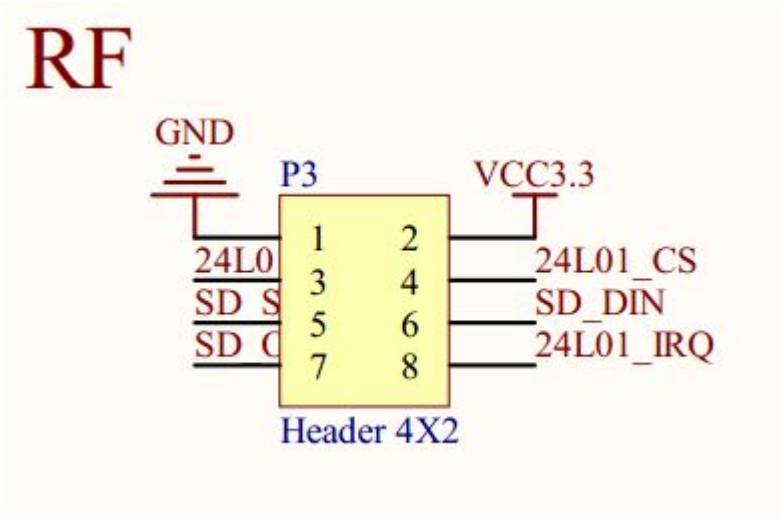
NRF24L01 是采用 spi 接口进行数据传输，使用起来也是比较简单的。无线鼠标或者无线键盘都可以采用这个芯片，一共具有 23 个寄存器。因此设计方法和之前讲的 spi 通信相类似。

2.21.2 硬件准备：

首先来看看：

```
//  * 硬件连接 ---------------------------
//  *          | PA8-CE          :24L01-CE |
//  *          | PC9-IRQ  :      24L01-IRQ |
//  *          | PC8-CS   :       24L01-CS  |
//  *          | PB13-SPI2-SCK  : 24L01-CLK |
//  *          | PB14-SPI2-MISO : 24L01-DO  |
//  *          | PB15-SPI2-MOSI : 24L01-DIO |
//  *           ---------------------------
```

硬件电路图设计如下：



2.21.3 软件设计

首先我们需要对 RF24L01 所使用到的 SPI 端口进行初始化，这是设计接口操作的第一步：包含：配置 IO 端口和配置 SPI 参数。

配置 IO 端口可以按下面方式配置：

```
01.    GPIO_InitTypeDef    GPIO_InitStruct;
02.    SPI_InitTypeDef     SPI_InitStruct;
03. RCC_AHBPeriphClockCmd(FLASH_CS_PIN_SCK|FLASH_SCK_PIN_SCK|FLASH_MISO_
    PIN_SCK | FLASH_MOSI_PIN_SCK, ENABLE);//设置时钟
04.    RCC_APB1PeriphClockCmd(RF_SPI2, ENABLE);
05.
06.    /*!< 配置  RF_SPI pins: SCK */
07.    GPIO_InitStruct.GPIO_Pin = RF_SCK_PIN;
08.    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
09.    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
10.    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
11.    GPIO_InitStruct.GPIO_PuPd    = GPIO_PuPd_UP;
12.    GPIO_Init(RF_SCK_PORT, &GPIO_InitStruct);
13.    /*!< 配置  RF_SPI pins: MISO */
14.    GPIO_InitStruct.GPIO_Pin = RF_MISO_PIN;
15.    GPIO_Init(RF_MISO_PORT, &GPIO_InitStruct);
16.    /*!<配置 RF_SPI pins: MOSI */
17.    GPIO_InitStruct.GPIO_Pin =RF_MOSI_PIN;
18.    GPIO_Init(RF_MOSI_PORT, &GPIO_InitStruct);
19.
20.    /*  设置 SPI 复用*/
21.    GPIO_PinAFConfig(RF_SCK_PORT, RF_SCK_SOURCE, RF_SCK_AF);
22.    GPIO_PinAFConfig(RF_MISO_PORT, RF_MISO_SOURCE, RF_MISO_AF);
23.    GPIO_PinAFConfig(RF_MOSI_PORT, RF_MOSI_SOURCE, RF_MOSI_AF);
24.
25.
```

```
26.    /*!< 配置 RF 的 CS pin */
27.     GPIO_InitStruct.GPIO_Pin =RF_CS_PIN;
28.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
29.     GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
30.     GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
31.     GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
32.     GPIO_Init(RF_CS_PORT, &GPIO_InitStruct);
33.
34.    GPIO_InitStruct.GPIO_Pin =RF_CE_PIN;
35.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
36.     GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
37.     GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
38.     GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
39.     GPIO_Init(RF_CE_PORT, &GPIO_InitStruct);
40.    SPI_RF_CS_HIGH() ;
41.    GPIO_InitStruct.GPIO_Pin =RF_IQR_PIN;
42.     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
43.     GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
44.     GPIO_InitStruct.GPIO_Speed = GPIO_Speed_Level_2;
45.     GPIO_Init(RF_IQR_PORT , &GPIO_InitStruct);
46.      SPI_RF_CE_LOW();
```

配置 SPI 的参数所有的库函数在前面使用 SPI 的时候已经详细讲解过，这里主要就是要注意设置 **SPI_CPOL** 为第电平有效和 **SPI_CPHA** 为第一边缘触发。

```
47.     /*!< SD_SPI Config */
48.     SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
49.     SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
50.     SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
51.    SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;          //时钟极性，空闲时为低
52.     SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;
53.     SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;
54.     SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
55.     SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
56.     SPI_InitStruct.SPI_CRCPolynomial = 7;
57.     SPI_Init(SPI2, &SPI_InitStruct);
58.     SPI_RxFIFOThresholdConfig(SPI2, SPI_RxFIFOThreshold_QF);
59.     SPI_Cmd(SPI2, ENABLE); /*!< SD_SPI enable */
```

那么 spi 给出几个寄存器命令：

```
/**********   NRF24L01 寄存器操作命令   ***********/
#define READ_REG          0x00   //读配置寄存器,低 5 位为寄存器地址
#define WRITE_REG         0x20   //写配置寄存器,低 5 位为寄存器地址
#define RD_RX_PLOAD       0x61   //读 RX 有效数据,1~32 字节
#define WR_TX_PLOAD       0xA0   //写 TX 有效数据,1~32 字节
#define FLUSH_TX          0xE1   //清除 TX FIFO 寄存器.发射模式下用
```

```
#define FLUSH_RX        0xE2    //清除 RX FIFO 寄存器.接收模式下用
#define REUSE_TX_PL     0xE3    //重新使用上一包数据,CE 为高,数据包被不断发送.
#define NOP             0xFF    //空操作,可以用来读状态寄存器
```

首先写一个寄存器：

```
/**************************************************/
/* 函数功能：给 24L01 的寄存器写值（一个字节） */
/* 入口参数：reg     要写的寄存器地址              */
/*           value  给寄存器写的值                */
/* 出口参数：status  状态值                       */
/**************************************************/
uint8_t NRF24L01_Write_Reg(uint8_t reg,uint8_t value)
{
 uint8_t status;

 SPI_RF_CS_LOW() ;   //CSN=0;
   status = SPI_RF_SendByte(reg);//发送寄存器地址,并读取状态值
 SPI_RF_SendByte(value);
 SPI_RF_CS_HIGH();    //CSN=1;
 return status;
}
```

读一个寄存器：

```
/**************************************************/
/* 函数功能：读 24L01 的寄存器值 （一个字节）      */
/* 入口参数：reg   要读的寄存器地址                */
/* 出口参数：value  读出寄存器的值                 */
/**************************************************/
uint8_t NRF24L01_Read_Reg(uint8_t reg)
{
      uint8_t value;

 SPI_RF_CS_LOW() ; //CSN=0;
   SPI_RF_SendByte(reg);//发送寄存器值(位置),并读取状态值
 value = SPI_RF_SendByte(NOP);
 SPI_RF_CS_HIGH();   //CSN=1;
 return value;
}
```

读多个寄存器：

```
/**************************************************/
/* 函数功能：读 24L01 的寄存器值（多个字节）     */
/* 入口参数：reg     寄存器地址                   */
/*           *pBuf  读出寄存器值的存放数组       */
/*           len     数组字节长度                 */
/* 出口参数：status  状态值                       */
/**************************************************/
```

```
98. uint8_t NRF24L01_Read_Buf(uint8_t reg,uint8_t *pBuf,uint8_t len)
99. {
100.      uint8_t status,u8_ctr;
101.      SPI_RF_CS_LOW() ;//CSN=0
102.   status=SPI_RF_SendByte(reg);//发送寄存器地址,并读取状态值
103.      for(u8_ctr=0;u8_ctr<len;u8_ctr++)
104.      pBuf[u8_ctr]=SPI_RF_SendByte(0XFF);//读出数据
105.      SPI_RF_CS_HIGH(); //CSN=1
106.   return status;          //返回读到的状态值
107. }
```

写多个寄存器：

```
108. /*********************************************/
109. /* 函数功能：给 24L01 的寄存器写值（多个字节）  */
110./* 入口参数：reg   要写的寄存器地址              */
111./*             *pBuf  值的存放数组               */
112./*             len    数组字节长度               */
113./*********************************************/
114.uint8_t NRF24L01_Write_Buf(uint8_t reg, uint8_t *pBuf, uint8_t len)
115.{
116. uint8_t status,u8_ctr;
117. SPI_RF_CS_LOW() ;
118.   status = SPI_RF_SendByte(reg);//发送寄存器值(位置),并读取状态值
119.   for(u8_ctr=0; u8_ctr<len; u8_ctr++)
120.      SPI_RF_SendByte(*pBuf++); //写入数据
121.      SPI_RF_CS_HIGH();
122.   return status;          //返回读到的状态值
123. }
```

设置 24L01 为接收模式：

```
124. /*********************************************/
125. /* 函数功能：设置 24L01 为接收模式              */
126. /*********************************************/
127. void NRF24L01_RX_Mode(void)
128. {
129.
130.      SPI_RF_CE_LOW() ; //CE 拉低，使能 24L01 配置
131.
132.      NRF24L01_Write_Buf(WRITE_REG+RX_ADDR_P0, (uint8_t*)RX_ADDRESS,
    RX_ADR_WIDTH);//写 RX 接收地址
133.
134.      NRF24L01_Write_Reg(WRITE_REG+EN_AA,0x01);     //开启通道 0 自动应答
135.      NRF24L01_Write_Reg(WRITE_REG+EN_RXADDR,0x01);//通道 0 接收允许
136.      NRF24L01_Write_Reg(WRITE_REG+RF_CH,40);          //设置 RF 工作通道频率
137.      NRF24L01_Write_Reg(WRITE_REG+RX_PW_P0,RX_PLOAD_WIDTH);
   / /选择通道 0 的有效数据宽度
```

```
138.      NRF24L01_Write_Reg(WRITE_REG+RF_SETUP,0x0f);
    //设置 TX 发射参数,0db 增益,2Mbps,低噪声增益开启
139.      NRF24L01_Write_Reg(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
140.      NRF24L01_Write_Reg(FLUSH_RX,0xff);//清除 RX FIFO 寄存器
141.      SPI_RF_CE_HIGH();  //CE 置高，使能接收
142. }
```

设置 24L01 为接收模式：

```
143. /*********************************************/
144. /*  函数功能：24L01 接收数据                        */
145. /*  入口参数：rxbuf  接收数据数组                  */
146. /*  返回值：  0     成功收到数据                    */
147. /*            1     没有收到数据                    */
148. /*********************************************/
149. uint8_t NRF24L01_RxPacket(uint8_t *rxbuf)
150. {
151.      uint8_t state;
152.
153.      state=NRF24L01_Read_Reg(STATUS);   //读取状态寄存器的值
154.      NRF24L01_Write_Reg(WRITE_REG+STATUS,state); //清除 TX_DS 或 MAX_RT 中断标
    志
155.      if(state&RX_OK)//接收到数据
156.      {
157.           NRF24L01_Read_Buf(RD_RX_PLOAD,rxbuf,RX_PLOAD_WIDTH);//读取数据
158.           NRF24L01_Write_Reg(FLUSH_RX,0xff);//清除 RX FIFO 寄存器
159.           return 0;
160.      }
161.      return 1;//没收到任何数据
162. }
```

24L01 接收数据：

```
163. /*********************************************/
164. /*  函数功能：24L01 接收数据                        */
165. /*  入口参数：rxbuf  接收数据数组                  */
166. /*  返回值：  0     成功收到数据                    */
167. /*            1     没有收到数据                    */
168. /*********************************************/
169. uint8_t NRF24L01_RxPacket(uint8_t *rxbuf)
170. {
171.      uint8_t state;
172.      state=NRF24L01_Read_Reg(STATUS);   //读取状态寄存器的值
173.      NRF24L01_Write_Reg(WRITE_REG+STATUS,state); //清除 TX_DS 或 MAX_RT 中断标
    志
174.      if(state&RX_OK)//接收到数据
175.      {
```

```
176.            NRF24L01_Read_Buf(RD_RX_PLOAD,rxbuf,RX_PLOAD_WIDTH);//读取数据
177.            NRF24L01_Write_Reg(FLUSH_RX,0xff);//清除 RX FIFO 寄存器
178.            return 0;
179.      }
180.      return 1;//没收到任何数据
181. }
```

设置 24L01 为发送模式：

```
182. /**********************************************/
183. /* 函数功能：设置 24L01 为发送模式                    */
184. /* 入口参数：txbuf  发送数据数组                     */
185. /* 返回值；  0x10    达到最大重发次数，发送失败*/
186. /*           0x20    成功发送完成                    */
187. /*           0xff    发送失败                        */
188. /**********************************************/
189. uint8_t NRF24L01_TxPacket(uint8_t *txbuf)
190. {
191.      uint8_t state;
192.      SPI_RF_CE_LOW() ; //CE 拉低，使能 24L01 配置
193.   NRF24L01_Write_Buf(WR_TX_PLOAD,txbuf,TX_PLOAD_WIDTH);//写数据到 TX BUF
    32 个字节
194.      SPI_RF_CE_HIGH(); //CE 置高，使能发送
195.      while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);//等待发送完成
196.      state=NRF24L01_Read_Reg(STATUS);  //读取状态寄存器的值
197.      NRF24L01_Write_Reg(WRITE_REG+STATUS,state); //清除 TX_DS 或 MAX_RT 中断标
    志
198.      if(state&MAX_TX)//达到最大重发次数
199.      {
200.          NRF24L01_Write_Reg(FLUSH_TX,0xff);//清除 TX FIFO 寄存器
201.          return MAX_TX;
202.      }
203.      if(state&TX_OK)//发送完成
204.      {
205.          return TX_OK;
206.      }
207.      return 0xff;//发送失败
208. }
```