JF24D 2.4G 无线传输测试

```
#define uchar unsigned char
#include"REG52.H"
#include"JF24D.h"
#include"pin_t.h"
extern inf_dat;
extern void    key();
extern void _ExternISR(void);
void CPU_Initialize();

/******************定义 KEY 和 LED 管脚*********************************/


 UINT8 pbuf[32]={""};            //发射的数据
 UINT8 pp[MAX_PACKET_LEN];      //中断接收的数据

 UINT8 GDataNumber=0x00;        //串口接收数据的位数
 UINT8 Ctr_=0x00;                //控制位  0 继续接收     FF 可以发送
 UINT8 INT_flg=0x00;

/*****************************************************************************
**函数名称：main()
**函数描述：
**入口参数：无
**出口参数：无
*****************************************************************************/
main()
{
      EA=0;                                          //初始化外部中断
      Delay(10);
      TMOD=0x02;
      TH0=0x00;
      TL0=0x00;
      ET0=1;
      EA=1;
      TR0=1;
      IT0=1;
      EX0=1;
      ES=1;
      while(1)
      {
       //Tx_byte(TH0);
```

```
        if(inf_dat==0x01)
        LED1=0;
    }
}
```

```
/*****************************************************************************
**函数名称：Delay()
**函数描述：延时
**入口参数：延时时间 毫秒
**出口参数：无
*****************************************************************************/
void Delay(UINT8 n)
{
  unsigned char i,m;
  for(i=0;i<n;i++)
    {
        for(m=0;m<220;m++);
        for(m=0;m<220;m++);
     }
}
/*****************************************************************************
**函数名称：void   Tx_byte(char a)
**函数描述：向串口缓冲区写一个字节
**入口参数：需要发送的字节
**出口参数：无
*****************************************************************************/

void   Tx_byte(char a)
{
    ES=0;
    TI=0;
    SBUF = a;
    do   ;
    while(TI==0) ;
    TI=0;
    ES=1;
}
```

```
/*********************************************************************
**函数名称：void uart(void) interrupt 4
**函数描述：串口中断函数
**入口参数：无
**出口参数：无
*********************************************************************/
void uart(void) interrupt 4
{
if(RI)
      {
        RI=0;
       if(GDataNumber>32)            //普通模式超过 32 个字节  给予提示
             {
             Tx_byte('F');
             Tx_byte('U');
             Tx_byte('L');
             Tx_byte('L');
             GDataNumber=0;
             }

       pbuf[GDataNumber]=SBUF;
       GDataNumber++;

       if(SBUF=='~')
            {
            Ctr_=0xff;                 //收到结束标志  可以开始发送
            }

      }
else
     TI=0;
}
 /*********************************************************************
**函数名称：void remot interrupt 2
**函数描述：红外接收中断函数
**入口参数：无
**出口参数：无
*********************************************************************/

 void EXTINT1() interrupt 0

 {

     // LED1=0;
```

```
    _ExternISR();


 }

/****************************************Copyright
(c)*************************************************
**                                          安阳新世纪电子研究所
**                                                技术部
**                                        http://www.ayxsj.com
**
**--------------File Info-----------------------------------------------------------------------
** File name:                JF24D.c
** Last modified Date:    2010-03-22
** Last Version:             1.0
** Descriptions:             Every project should include a copy of this file.
**
**-------------------------------------------------------------------------------------------------
** Created by:               lizhiyuan
** Created date:             2010-03-22
** Version:                      1.0
** Descriptions:             The original version
**
**-------------------------------------------------------------------------------------------------
** Modified by:
** Modified date:
** Version:
** Descriptions:
**
************************************************************************************
*************************/
#include"REG52.H"
#include"JF24D.h"
#include"pin_t.h"



//寄存器组 1    0-13 号寄存器的初始值
code UINT32 RegArrFSKAnalog[]={
0xF2014B41,
0x30064BC0,
0x00C4FCA0,
0x60350017,
0x0B009941,     // 0B109941 for 2 Mbps mode
```

```
0xBE7F0124,
0x00400000,
0x00000000,
0x00000000,
0x00000000,
0xF64EF5F6,
0x5C1851D6,
0x4055002D,
0x00700000        // 00040000 for 2 Mbps mode
};

//寄存器组 1    14 号寄存器的初始值
code UINT8 RegArrFSKAnalogReg14[]=
{
0x41,0x10,0x08,0x82,0x40,0x10,0x08,0xF2,0x7C,0xEF,0xCF
};

//寄存器组 0 初始值
code UINT8 RegArrFSK[][2]={
{0,0x0F},              //配置寄存器
{1,0x3F},
{2,0x3F},              //使能接收地址
{3,0x03},        //信道的数据长度
{4,0xff},
{5,0x17},             //频道选择寄存器
{6,0x17},
{7,0x07},        //状态寄存器
{8,0x00},        //射频设置寄存器
{9,0x00},
{12,0xc3},
{13,0xc4},
{14,0xc5},
{15,0xc6},
{17,0x20},
{18,0x20},
{19,0x20},
{20,0x20},
{21,0x20},
{22,0x20},
{23,0x00},
{28,0x3F},
{29,0x07}
};
```

```
code UINT8 RX0_Address[]={0x12,0x34,0x56,0x78,0x01};      //接收通道 0 的地址
code UINT8 RX1_Address[]={0xc2,0x56,0x34,0x12,0x02};      //接收通道 1 的地址

/*************************************************************************
******
**函数名称：SPI_RW()
**函数描述：写一个字节到 JF24D，并返回读出的字节
**          平时低电平，上升沿读数据,下降沿写数据
**入口参数：命令或地址
**出口参数：读出的字节
*************************************************************************
*****/
UINT8 SPI_RW(UINT8 _byte)
{
    UINT8 bit_ctr;
    for(bit_ctr=0;bit_ctr<8;bit_ctr++)
    {
        MOSI = (_byte & 0x80);          // 输出，先输出高位
        _byte = (_byte << 1);           // 下一位输出值移位到高位
        SCK = 1;                        // SCK 置位
        _byte |= MISO;                  // 读 MISO 当前值
        SCK = 0;                        // SCK 清零
    }
    return(_byte);                      // 返回读出的值
}

/*************************************************************
**函数名称：SPI_Write_Reg()
**函数描述：写寄存器的值
**入口参数：寄存器地址+命令，寄存器的值
**出口参数：无
*************************************************************/
void SPI_Write_Reg(UINT8 reg, UINT8 value)
{
    CSN = 0;                        // 清零 CSN，使能 SPI
    SPI_RW(reg);                    // 写寄存器地址+命令
    SPI_RW(value);                  // 写相应的值
    CSN = 1;                        // 置位 CSN，禁止 SPI
}

/*************************************************************
**函数名称：SPI_Read_Reg()
**函数描述：读寄存器的值
**入口参数：寄存器地址+命令
```

```
**出口参数：寄存器的值
*************************************************************/
UINT8 SPI_Read_Reg(UINT8 reg)
{
    BYTE value;
    CSN = 0;                    // 清零 CSN，使能 SPI
    SPI_RW(reg);                // 写寄存器地址+命令
    value = SPI_RW(0);          // 读寄存器的值
    CSN = 1;                    // 置位 CSN，禁止 SPI

    return(value);              // 返回寄存器的值
}

/*************************************************************
**函数名称：SPI_Read_Buf()
**函数描述：读多字节寄存器的值
**入口参数：寄存器地址+命令，返回值的地址，寄存器值的长度
**出口参数：无
*************************************************************/
void SPI_Read_Buf(UINT8 reg, UINT8 *pBuf, UINT8 bytes)
{
    UINT8 byte_ctr;

    CSN = 0;                            // 清零 CSN，使能 SPI
    SPI_RW(reg);                // 写寄存器地址+命令

    for(byte_ctr=0;byte_ctr<bytes;byte_ctr++)
        pBuf[byte_ctr] = SPI_RW(0);     // 读寄存器的值

    CSN = 1;                                // 置位 CSN，禁止 SPI
}

/*************************************************************
**函数名称：SPI_Write_Buf()
**函数描述：写多字节寄存器的值
**入口参数：寄存器地址+命令，写入值的地址，寄存器值的长度
**出口参数：无
*************************************************************/
void SPI_Write_Buf(UINT8 reg, UINT8 *pBuf, UINT8 bytes) reentrant
{
    UINT8 byte_ctr;

    CSN = 0;                            // 清零 CSN，使能 SPI
    SPI_RW(reg);      // 写寄存器地址+命令
```

```
    for(byte_ctr=0; byte_ctr<bytes; byte_ctr++) //  写寄存器的值
        SPI_RW(*(pBuf+byte_ctr));
    CSN = 1;                           //  置位 CSN，禁止 SPI
}

/*********************************************************
**函数名称：Enter_PRX()
**函数描述：切换到 PRX 模式
**入口参数：无
**出口参数：无
*********************************************************/
void Enter_PRX()
{
    UINT8 value;

    SPI_Write_Reg(FLUSH_RX,0);                    //清空接收 FIFO

    value=SPI_Read_Reg(STATUS);                   //  读 STATUS 寄存器
    SPI_Write_Reg(WRITE_REG+STATUS,value);        //  清零 RX_DR、TX_DS、MAX_RT
标志

    value=SPI_Read_Reg(CONFIG);                   //  读 CONFIG 寄存器

    value=value&0xfd;                             //位 1 清零
    SPI_Write_Reg(WRITE_REG + CONFIG, value); //  清零 PWR_UP 位，进入
POWER_DOWN 模式

    value=value|0x03;                             //置位位 1，位 0
    SPI_Write_Reg(WRITE_REG + CONFIG, value); //  置位 PWR_UP，PRIM_RX，进入 PRX
模式
}

/*********************************************************
**函数名称：Enter_PTX()
**函数描述：切换到 PTX 模式
**入口参数：无
**出口参数：无
*********************************************************/
void Enter_PTX()
{
    UINT8 value;

    SPI_Write_Reg(FLUSH_TX,0);                    //清空接收 FIFO
```

```
    value=SPI_Read_Reg(STATUS);                    // 读 STATUS 寄存器
    SPI_Write_Reg(WRITE_REG+STATUS,value); // 清零 RX_DR、TX_DS、MAX_RT 标志

    value=SPI_Read_Reg(CONFIG);                    // 读 CONFIG 寄存器
    value=value&0xfd;                              //位 1 清零
    SPI_Write_Reg(WRITE_REG+CONFIG,  value);//  清  零  PWR_UP  位  ，  进  入
POWER_DOWN 模式

    value=value|0x02;                              //置位位 1
    value=value&0xfe;                              //位 0 清零
    SPI_Write_Reg(WRITE_REG + CONFIG, value); // 置位 PWR_UP，清零 PRIM_RX，进
入 PTX 模式
}

/****************************************************************
**函数名称：SwitchCFG()
**函数描述：切换寄存器组
**入口参数：将要切换到寄存器组，0 或 1
**出口参数：无
****************************************************************/
void SwitchCFG(char _cfg)
{
    UINT8 Tmp;

    Tmp=SPI_Read_Reg(STATUS);         //读 STATUS 寄存器

    Tmp=Tmp&0x80;

    if( ( Tmp&&(_cfg==0) )
    ||( (Tmp==0)&&_cfg ) )              //判断当前寄存器组是否是将要切换的
    {
        SPI_Write_Reg(ACTIVATE_CMD,0x53);              //执行切换
    }
}

/****************************************************************
**函数名称：Send_Packet()
**函数描述：发射数据包
**入口参数：写发射 FIFO 命令，写入值的地址，寄存器值的长度
**出口参数：无
****************************************************************/
void Send_Packet(UINT8 type,UINT8* pbuf,UINT8 len)
{
    UINT8 fifo_sta;
```

```
    fifo_sta=SPI_Read_Reg(FIFO_STATUS);       // 读寄存器 FIFO_STATUS

    if((fifo_sta&FIFO_STATUS_TX_FULL)==0) //判断发射 FIFO 是否满
        SPI_Write_Buf(type, pbuf, len);     // 写发射 FIFO
}

/****************************************************************
**函数名称：Receive_Packet()
**函数描述：接收数据包
**入口参数：读接收 FIFO 命令，读出值的地址
**出口参数：数据包的长度
****************************************************************/
UINT8 Receive_Packet(UINT8 rx_buf[MAX_PACKET_LEN])
{
    UINT8 len,fifo_sta;

    do
    {
        len=SPI_Read_Reg(R_RX_PL_WID_CMD);     // 读数据包长度

        if(len<=MAX_PACKET_LEN)
        {
                SPI_Read_Buf(RD_RX_PLOAD,rx_buf,len);// 读接收 FIFO
        }
        else
        {
            SPI_Write_Reg(FLUSH_RX,0);          //数据包长度大于最大长度，清空 FIFO
        }

        fifo_sta=SPI_Read_Reg(FIFO_STATUS);   //读寄存器 FIFO_STATUS

    }while((fifo_sta&FIFO_STATUS_RX_EMPTY)==0); //如果不为空，继续读

    return(len);                                //返回数据包长度
}

/****************************************************************
**函数名称：JF24D_Init()
**函数描述：初始化 JF24D
**入口参数：无
**出口参数：无
****************************************************************/
void JF24D_Init()
```

```
{
    INT8 i,j;
    UINT8 WriteArr[4];

/************************初始化寄存器组 1*********************************/
    SwitchCFG(1);        //切换到寄存器组 1

    for(i=0;i<=8;i++)              //写前 0-8 号寄存器
    {
        for(j=0;j<4;j++)
        {
            WriteArr[j]=(RegArrFSKAnalog[i]>>(8*(j) ) )&0xff;   //将寄存器值存放到数组
中，先高字节
        }
            SPI_Write_Buf((WRITE_REG|i),&(WriteArr[0]),4);        //写寄存器
    }

    for(i=9;i<=13;i++)                                              //写 9-13 号寄存
器
    {
        for(j=0;j<4;j++)
        {
        WriteArr[j]=(RegArrFSKAnalog[i]>>(8*(3-j) ) )&0xff;     //将寄存器值存放到数组
中，先低字节
        }
        SPI_Write_Buf((WRITE_REG|i),&(WriteArr[0]),4);              //写寄存器
    }

    SPI_Write_Buf((WRITE_REG|14),&(RegArrFSKAnalogReg14[0]),11);  //写 14 号寄存器

/***************写 REG4[0]=0，REG4[1]=0，REG4[3]=0，REG4[23]=0 **************/
    for(j=0;j<4;j++)
    {
        WriteArr[j]=(RegArrFSKAnalog[4]>>(8*(j) ) )&0xff;
    }
    WriteArr[3]=WriteArr[3]&0xf4;
    WriteArr[1]=WriteArr[1]&0x7F;
    SPI_Write_Buf((WRITE_REG|4),&(WriteArr[0]),4);

/************写 REG4[25]=1，写 REG4[26]=1 *********************************/
    WriteArr[0]=WriteArr[0]|0x06;
    SPI_Write_Buf((WRITE_REG|4),&(WriteArr[0]),4);

/*************写 REG4[25]=0，写 REG4[26]=0*****************************/
```

```
    WriteArr[0]=WriteArr[0]&0xf9;
    SPI_Write_Buf((WRITE_REG|4),&(WriteArr[0]),4);

/************写 REG4[0]=1，REG4[1]=1，REG4[3]=1，REG4[23]=1**********/
    for(j=0;j<4;j++)
    {
        WriteArr[j]=(RegArrFSKAnalog[4]>>(8*(j) ) )&0xff;
    }
    SPI_Write_Buf((WRITE_REG|4),&(WriteArr[0]),4);

/******************初始化寄存器组 0**********************************/
    SwitchCFG(0);           //切换到寄存器组 0

    for(i=20;i>=0;i--)
    {
        SPI_Write_Reg((WRITE_REG|RegArrFSK[i][0]),RegArrFSK[i][1]);     //初始化前面
定义的寄存器
    }

    SPI_Write_Buf((WRITE_REG+RX_ADDR_P0),RX0_Address,5);        //写寄存器 10，通
道 0 地址                寄存器地址+命令，返回值的地址，寄存器值的长度

    SPI_Write_Buf((WRITE_REG+RX_ADDR_P1),RX1_Address,5);        //写寄存器 11，通
道 1 地址，及其余通道高位地址

    SPI_Write_Buf((WRITE_REG+TX_ADDR),RX0_Address,5);           //写寄存器 16，发
射通道地址

    i=SPI_Read_Reg(29);         //读寄存器 29
    if(i==0)            //是否需要激活
    {
        SPI_Write_Reg(ACTIVATE_CMD,0x73);   //激活寄存器 29
            for(i=22;i>=21;i--)
            {
                SPI_Write_Reg((WRITE_REG|RegArrFSK[i][0]),RegArrFSK[i][1]);     //
写寄存器 28、29
            }
    }

/************************发射连续波 CW*****************************/
#ifdef TX_CW
    SwitchCFG(1);
     WriteArr[0] = 0x41;
     WriteArr[1] = 0x11;
```

```
        WriteArr[2] = 0x04;
        WriteArr[3] = 0x21;
            SPI_Write_Buf((WRITE_REG|4),&(WriteArr[0]),4);
        while(1);
#endif
}


#ifndef __JF24D__
#define __JF24D__


#include "intrins.h"

#define BYTE unsigned char
#define INT8    unsigned char
#define INT16 int
#define UINT8 unsigned char
#define UINT16 unsigned int
#define UINT32 unsigned long
#define MAX_PACKET_LEN    32// 数据包最大长度，最大为 255

/***********SPI 命令*********************************/
#define READ_REG            0x00            // 读寄存器
#define WRITE_REG           0x20            // 写寄存器
#define RD_RX_PLOAD         0x61             // 读接收 FIFO
#define WR_TX_PLOAD         0xA0             // 写发射 FIFO
#define FLUSH_TX            0xE1            // 清空发射 FIFO
#define FLUSH_RX            0xE2            // 清空接收 FIFO
#define REUSE_TX_PL         0xE3            // 重复发射数据包
#define W_TX_PAYLOAD_NOACK_CMD0xb0        //写发射 FIFO，无应答
#define W_ACK_PAYLOAD_CMD     0xa8  //写应答 FIFO
#define ACTIVATE_CMD          0x50        //ACTIVATE 命令
#define R_RX_PL_WID_CMD         0x60        //读数据包长度
#define NOP                   0xFF    // 无操作

/*********寄存器组 0 的寄存器声明**********************/
#define CONFIG              0x00    // 'Config' register address
#define EN_AA               0x01    // 'Enable Auto Acknowledgment' register address
#define EN_RXADDR           0x02    // 'Enabled RX addresses' register address
#define SETUP_AW            0x03    // 'Setup address width' register address
#define SETUP_RETR          0x04    // 'Setup Auto. Retrans' register address
#define RF_CH               0x05    // 'RF channel' register address
#define RF_SETUP            0x06    // 'RF setup' register address
```

```
#define STATUS               0x07    // 'Status' register address
#define OBSERVE_TX           0x08    // 'Observe TX' register address
#define CD                   0x09    // 'Carrier Detect' register address
#define RX_ADDR_P0           0x0A    // 'RX address pipe0' register address
#define RX_ADDR_P1           0x0B    // 'RX address pipe1' register address
#define RX_ADDR_P2           0x0C    // 'RX address pipe2' register address
#define RX_ADDR_P3           0x0D    // 'RX address pipe3' register address
#define RX_ADDR_P4           0x0E    // 'RX address pipe4' register address
#define RX_ADDR_P5           0x0F    // 'RX address pipe5' register address
#define TX_ADDR              0x10    // 'TX address' register address
#define RX_PW_P0             0x11    // 'RX payload width, pipe0' register address
#define RX_PW_P1             0x12    // 'RX payload width, pipe1' register address
#define RX_PW_P2             0x13    // 'RX payload width, pipe2' register address
#define RX_PW_P3             0x14    // 'RX payload width, pipe3' register address
#define RX_PW_P4             0x15    // 'RX payload width, pipe4' register address
#define RX_PW_P5             0x16    // 'RX payload width, pipe5' register address
#define FIFO_STATUS          0x17    // 'FIFO Status Register' register address
#define PAYLOAD_WIDTH        0x1f    // 'payload length of 256 bytes modes register address

/************STATUS 寄存器状态*******************/
#define STATUS_RX_DR 0x40
#define STATUS_TX_DS 0x20
#define STATUS_MAX_RT 0x10
#define STATUS_TX_FULL 0x01

/***********FIFO_STATUS 寄存器状态**********************/
#define FIFO_STATUS_TX_REUSE 0x40
#define FIFO_STATUS_TX_FULL 0x20
#define FIFO_STATUS_TX_EMPTY 0x10
#define FIFO_STATUS_RX_FULL 0x02
#define FIFO_STATUS_RX_EMPTY 0x01

#define ENTER_TX_RX() CE=1       //进入接收或发射
#define EXIT_TX_RX()    CE=0       //退出接收或发射

/***************使用函数声明*******************************/
UINT8 SPI_Read_Reg(UINT8 reg);
void SPI_Read_Buf(UINT8 reg, UINT8 *pBuf, UINT8 bytes);
void SPI_Write_Reg(UINT8 reg, UINT8 value);
void SPI_Write_Buf(UINT8 reg, UINT8 *pBuf, UINT8 bytes) reentrant;

void SwitchCFG(char _cfg);

void Delay(UINT8 n);
```

```
void JF24D_Init();
void Enter_PTX();
void Enter_PRX();
UINT8 Receive_Packet(UINT8 rx_buf[MAX_PACKET_LEN]);
void Send_Packet(UINT8 type,UINT8* pbuf,UINT8 len);
#endif



/*************************************************************************************
***************************
**                                  End Of File
*************************************************************************************
**************************/

/*************************************************************************************
****************
                                        END FILE
*************************************************************************************
***************/
```