http://blog.163.com/jammy_lee/

http://21eos.taobao.com

Linux 下驱动 2.4G 无线模块(NRF24L01)

linux 驱动 2010-02-06 11:29:45 阅读 613 评论 6 字号：大中小 订阅

NRF24L01 使用的 SPI 协议通信，这里并没有用到 s3c2440 自带的 SPI 功能，而是直接用 IO 口模拟 SPI。而根据 Datasheet 所示，NRF24L01 的 SPI 接速率为 0~8Mbps，因此需要在读写时序上加上适当的延时。

驱动程序：

```
/*************************************************************/
//文件名：nrf24l01.c
//功能:linux 下的 nrf24l01 驱动程序
//使用说明: (1)
//          (2)
//          (3)
//          (4)
//作者:jammy-lee
//日期：2010-01-11
/*************************************************************/

#include <linux/config.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/delay.h>
```

```
#include <linux/miscdevice.h>
#include <linux/devfs_fs_kernel.h>

#include <asm/uaccess.h>
#include <asm/hardware.h>
#include <asm/arch/regs-gpio.h>

typedef unsigned int uint16 ;
typedef unsigned char uint8 ;

/*
//和引脚相关的宏定义
#define CE         S3C2410_GPF3
#define CE_OUTP         S3C2410_GPF3_OUTP
#define SCK        S3C2410_GPF4
#define SCK_OUTP    S3C2410_GPF4_OUTP
#define MISO       S3C2410_GPG3
#define MISO_INP       S3C2410_GPG3_INP
#define IRQ        S3C2410_GPG0
#define IRQ_OUTP        S3C2410_GPG0_OUTP
#define MOSI       S3C2410_GPG5
#define MOSI_OUTP       S3C2410_GPG5_OUTP
#define CSN        S3C2410_GPG6
#define CSN_OUTP       S3C2410_GPG6_OUTP
*/

//和引脚相关的宏定义
#define CSN        S3C2410_GPF3
#define CSN_OUTP       S3C2410_GPF3_OUTP
#define MOSI       S3C2410_GPF4
#define MOSI_OUTP       S3C2410_GPF4_OUTP
#define IRQ        S3C2410_GPG3
```

```
#define IRQ_INP        S3C2410_GPG3_INP
#define MISO       S3C2410_GPG0
#define MISO_INP       S3C2410_GPG0_INP
#define SCK        S3C2410_GPG5
#define SCK_OUTP    S3C2410_GPG5_OUTP
#define CE         S3C2410_GPG6
#define CE_OUTP         S3C2410_GPG6_OUTP

#define DEVICE_NAME     "NRF24L01" //设备名称，在可以 /proc/devices 查看
#define NRF24L01_MAJOR    241  //主设备号
#define TxBufSize     32

uint8   TxBuf[TxBufSize]={
 0x01,0x02,0x03,0x4,0x05,0x06,0x07,0x08,
 0x09,0x10,0x11,0x12,0x13,0x14,0x15,0x16,
 0x17,0x18,0x19,0x20,0x21,0x22,0x23,0x24,
 0x25,0x26,0x27,0x28,0x29,0x30,0x31,0x32,
};

//NRF24L01 端口定义
#define CE_OUT  s3c2410_gpio_cfgpin(CE, CE_OUTP)  //数据线设置为输出
#define CE_UP         s3c2410_gpio_pullup(CE, 1)        //打开上拉电阻
#define CE_L  s3c2410_gpio_setpin(CE, 0)   //拉低数据线电平
#define CE_H  s3c2410_gpio_setpin(CE, 1)   //拉高数据线电平

#define SCK_OUT  s3c2410_gpio_cfgpin(SCK, SCK_OUTP) //数据线设置为输出
#define SCK_UP         s3c2410_gpio_pullup(SCK, 1)        //打开上拉电阻
#define SCK_L  s3c2410_gpio_setpin(SCK, 0)  //拉低数据线电平
#define SCK_H  s3c2410_gpio_setpin(SCK, 1)  //拉高数据线电平

#define MISO_IN  s3c2410_gpio_cfgpin(MISO, MISO_INP) //数据线设置为输出
#define MISO_UP         s3c2410_gpio_pullup(MISO, 1)        //打开上拉电阻
#define MISO_STU        s3c2410_gpio_getpin(MISO)   //数据状态
```

```
#define IRQ_IN  s3c2410_gpio_cfgpin(IRQ, IRQ_INP) //数据线设置为输出
#define IRQ_UP        s3c2410_gpio_pullup(IRQ, 1)       //打开上拉电阻
#define IRQ_L  s3c2410_gpio_setpin(IRQ, 0)   //拉低数据线电平
#define IRQ_H  s3c2410_gpio_setpin(IRQ, 1)   //拉高数据线电平

#define MOSI_OUT s3c2410_gpio_cfgpin(MOSI, MOSI_OUTP) //数据线设置为输出
#define MOSI_UP        s3c2410_gpio_pullup(MOSI, 1)       //打开上拉电阻
#define MOSI_L  s3c2410_gpio_setpin(MOSI, 0)  //拉低数据线电平
#define MOSI_H  s3c2410_gpio_setpin(MOSI, 1)  //拉高数据线电平

#define CSN_OUT  s3c2410_gpio_cfgpin(CSN, CSN_OUTP) //数据线设置为输出
#define CSN_UP        s3c2410_gpio_pullup(CSN, 1)       //打开上拉电阻
#define CSN_L  s3c2410_gpio_setpin(CSN, 0)  //拉低数据线电平
#define CSN_H  s3c2410_gpio_setpin(CSN, 1)  //拉高数据线电平


//NRF24L01
#define TX_ADR_WIDTH    5        // 5 uint8s TX address width
#define RX_ADR_WIDTH    5        // 5 uint8s RX address width
#define TX_PLOAD_WIDTH  32    // 20 uint8s TX payload
#define RX_PLOAD_WIDTH  32       // 20 uint8s TX payload
uint8 const TX_ADDRESS[TX_ADR_WIDTH]= {0x34,0x43,0x10,0x10,0x01};    //本地地址
uint8 const RX_ADDRESS[RX_ADR_WIDTH]= {0x34,0x43,0x10,0x10,0x01};    //接收地址


//NRF24L01 寄存器指令
#define READ_REG        0x00    //  读寄存器指令
#define WRITE_REG       0x20    //  写寄存器指令
#define RD_RX_PLOAD     0x61    //  读取接收数据指令
#define WR_TX_PLOAD     0xA0    //  写待发数据指令
#define FLUSH_TX        0xE1    //  冲洗发送  FIFO 指令
#define FLUSH_RX        0xE2    //  冲洗接收  FIFO 指令
```

```
#define REUSE_TX_PL     0xE3    // 定义重复装载数据指令
#define NOP             0xFF    // 保留

//SPI(nRF24L01)寄存器地址
#define CONFIG          0x00  // 配置收发状态，CRC 校验模式以及收发状态响应方式
#define EN_AA           0x01  // 自动应答功能设置
#define EN_RXADDR       0x02  // 可用信道设置
#define SETUP_AW        0x03  // 收发地址宽度设置
#define SETUP_RETR      0x04  // 自动重发功能设置
#define RF_CH           0x05  // 工作频率设置
#define RF_SETUP        0x06  // 发射速率、功耗功能设置
#define STATUS          0x07  // 状态寄存器
#define OBSERVE_TX      0x08  // 发送监测功能
#define CD              0x09  // 地址检测
#define RX_ADDR_P0      0x0A  // 频道 0 接收数据地址
#define RX_ADDR_P1      0x0B  // 频道 1 接收数据地址
#define RX_ADDR_P2      0x0C  // 频道 2 接收数据地址
#define RX_ADDR_P3      0x0D  // 频道 3 接收数据地址
#define RX_ADDR_P4      0x0E  // 频道 4 接收数据地址
#define RX_ADDR_P5      0x0F  // 频道 5 接收数据地址
#define TX_ADDR         0x10  // 发送地址寄存器
#define RX_PW_P0        0x11  // 接收频道 0 接收数据长度
#define RX_PW_P1        0x12  // 接收频道 0 接收数据长度
#define RX_PW_P2        0x13  // 接收频道 0 接收数据长度
#define RX_PW_P3        0x14  // 接收频道 0 接收数据长度
#define RX_PW_P4        0x15  // 接收频道 0 接收数据长度
#define RX_PW_P5        0x16  // 接收频道 0 接收数据长度
#define FIFO_STATUS     0x17  // FIFO 栈入栈出状态寄存器设置
```

```
uint8 init_NRF24L01(void);

uint8 SPI_RW(uint8 tmp);

uint8 SPI_Read(uint8 reg);

void SetRX_Mode(void);

uint8 SPI_RW_Reg(uint8 reg, uint8 value);

uint8 SPI_Read_Buf(uint8 reg, uint8 *pBuf, uint8 uchars);

uint8 SPI_Write_Buf(uint8 reg, uint8 *pBuf, uint8 uchars);

unsigned char nRF24L01_RxPacket(unsigned char* rx_buf);

void nRF24L01_TxPacket(unsigned char * tx_buf);

//全局变量

uint8 opencount = 0;


//

uint8      sta;    //状态标志

#define    RX_DR     6

#define    TX_DS     5

#define    MAX_RT    4


//NRF24L01 初始化

uint8 init_NRF24L01(void)

{

/*

    CE_UP;

    SCK_UP;

    MISO_UP;

    IRQ_UP;
```

```
    MOSI_UP;
    CSN_UP;
*/
    MISO_UP;
    CE_OUT;
    CSN_OUT;
    SCK_OUT;
    MOSI_OUT;
    MISO_IN;
    IRQ_IN;
    udelay(500);
    CE_L;    // chip enable
    ndelay(60);
    CSN_H;   // Spi disable
    ndelay(60);
    SCK_L;   // Spi clock line init high
    ndelay(60);
    SPI_Write_Buf(WRITE_REG + TX_ADDR, TX_ADDRESS, TX_ADR_WIDTH);    // 写本地地址
    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, RX_ADDRESS, RX_ADR_WIDTH); // 写接收端地址
    SPI_RW_Reg(WRITE_REG + EN_AA, 0x01);      //  频道 0 自动 ACK 应答允许
    SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);  //  允许接收地址只有频道 0，如果需要多频道可以参考 Page21
    SPI_RW_Reg(WRITE_REG + RF_CH, 0);         //  设置信道工作为 2.4GHZ，收发必须一致
    SPI_RW_Reg(WRITE_REG + RX_PW_P0, RX_PLOAD_WIDTH); //设置接收数据长度，本次设置为 32 字节
    SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);       //设置发射速率为 1MHZ，发射功率为最大值 0dB
    SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);         // IRQ 收发完成中断响应，16 位 CRC ，主接收
```

```
    mdelay(1000);
    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("test 1 \n");
    mdelay(1000);
/*
    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("test 2 \n");
    mdelay(1000);
    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("test 3 \n");
    mdelay(1000);
    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("test 4 \n");
    mdelay(1000);
*/
    return (1);
}

//函数：uint8 SPI_RW(uint8 tmp)
//功能：NRF24L01 的 SPI 写时序 tmp
uint8 SPI_RW(uint8 tmp)
{
    uint8 bit_ctr;

    for(bit_ctr=0 ;bit_ctr<8 ;bit_ctr++) // output 8-bit
    {
```

```
 if(tmp & 0x80)          // output 'tmp', MSB to MOSI
     MOSI_H;
 else
     MOSI_L;

         tmp <<= 1;            // shift next bit into MSB..
         SCK_H;                  // Set SCK high..
 ndelay(60);
         tmp |= MISO_STU;          // capture current MISO bit
         SCK_L;                  // ..then set SCK low again
 ndelay(60);
    }
    return(tmp);                  // return read tmp
}


//函数：uint8 SPI_Read(uint8 reg)
//功能：NRF24L01 的 SPI 时序
uint8 SPI_Read(uint8 reg)
{
    uint8 reg_val;
    CSN_L;                // CSN low, initialize SPI communication...
    ndelay(60);
    SPI_RW(reg);              // Select register to read from..
    reg_val = SPI_RW(0);      // ..then read registervalue
    CSN_H;                // CSN high, terminate SPI communication
    ndelay(60);

    return(reg_val);          // return register value
```

```
}


//功能：NRF24L01 读写寄存器函数
uint8 SPI_RW_Reg(uint8 reg, uint8 value)
{
    uint8 status;

    CSN_L;                   // CSN low, init SPI transaction
    ndelay(60);
    status = SPI_RW(reg);       // select register
    SPI_RW(value);              // ..and write value to it..
    CSN_H;                   // CSN high again
    ndelay(60);

    return(status);            // return nRF24L01 status uint8
}


//函数：uint8 SPI_Read_Buf(uint8 reg, uint8 *pBuf, uint8 uchars)
//功能: 用于读数据，reg：为寄存器地址，pBuf：为待读出数据地址，uchars：读出数据的个数
uint8 SPI_Read_Buf(uint8 reg, uint8 *pBuf, uint8 uchars)
{
    uint8 status,uint8_ctr;

    CSN_L;                         // Set CSN low, init SPI tranaction
    ndelay(60);
    status = SPI_RW(reg);               // Select register to write to and read status uint8

    for(uint8_ctr=0;uint8_ctr<uchars;uint8_ctr++)
    {
```

```
        pBuf[uint8_ctr] = SPI_RW(0);    //
        ndelay(20);
    }

    CSN_H;
    ndelay(60);

    return(status);                 // return nRF24L01 status uint8
}

//函数：uint8 SPI_Write_Buf(uint8 reg, uint8 *pBuf, uint8 uchars)
//功能: 用于写数据：为寄存器地址，pBuf：为待写入数据地址，uchars：写入数据的个数
uint8 SPI_Write_Buf(uint8 reg, uint8 *pBuf, uint8 uchars)
{
    uint8 status,uint8_ctr;

    CSN_L;          //SPI 使能
    ndelay(60);
    status = SPI_RW(reg);
    for(uint8_ctr=0; uint8_ctr<uchars; uint8_ctr++) //
    {
        SPI_RW(*pBuf++);
     ndelay(20);
    }
    CSN_H;          //关闭 SPI
    ndelay(60);
    return(status);    //
}
```

```
//函数：void SetRX_Mode(void)
//功能：数据接收配置
void SetRX_Mode(void)
{
    CE_L;
    ndelay(60);
//  SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);          // IRQ 收发完成中断响应，16 位 CRC ，主接收
    //udelay(1);
    CE_H;
    udelay(130);
}


//函数：unsigned char nRF24L01_RxPacket(unsigned char* rx_buf)
//功能：数据读取后放如 rx_buf 接收缓冲区中
unsigned char nRF24L01_RxPacket(unsigned char* rx_buf)
{
    unsigned char revale=0;

    sta=SPI_Read(STATUS);   // 读取状态寄存其来判断数据接收状况
    if(sta & (1<<RX_DR))     // 判断是否接收到数据
    {
       CE_L;            //SPI 使能
     udelay(50);
        SPI_Read_Buf(RD_RX_PLOAD,rx_buf,TX_PLOAD_WIDTH);// read receive payload from
RX_FIFO buffer
       revale =1;          //读取数据完成标志
    }
    SPI_RW_Reg(WRITE_REG+STATUS,sta);   //接收到数据后 RX_DR,TX_DS,MAX_PT 都置高为 1，通
过写 1 来清楚中断标志
```

```
    return revale;
}


//函数：void nRF24L01_TxPacket(unsigned char * tx_buf)
//功能：发送 tx_buf 中数据
void nRF24L01_TxPacket(unsigned char * tx_buf)
{
    CE_L;           //StandBy I 模式
    ndelay(60);
    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH); // 装载接收端地
址
    SPI_Write_Buf(WR_TX_PLOAD, tx_buf, TX_PLOAD_WIDTH);             // 装载数据
    SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);           // IRQ 收发完成中断响应，16 位 CRC，主发送
    CE_H;       //置高 CE，激发数据发送
    udelay(10);
}

//文件的写函数
static ssize_t nrf24l01_write(struct file *filp, const char *buffer,
            size_t count, loff_t *ppos)
{
    if( copy_from_user( &TxBuf, buffer, count ) );    //从内核空间复制到用户空间
    {
 printk("Can't Send Data !");
 return -EFAULT;
    }

    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("OK! \n");
    return(10);
```

```
}

//的读函数
static ssize_t nrf24l01_read(struct file *filp, char *buffer,
                size_t count, loff_t *ppos)
{
    nRF24L01_TxPacket(TxBuf);
    SPI_RW_Reg(WRITE_REG+STATUS,0XFF);
    printk("read \n");

    return (10);
}

static int nrf24l01_open(struct inode *node, struct file *file)
{
  uint8 flag = 0;

  if(opencount == 1)
    return -EBUSY;


  flag = init_NRF24L01();

  mdelay(100);
  if(flag == 0)
    {
      printk("uable to open device!\n");
      return -1;
    }
  else
   {
      opencount++;
      printk("device opened !\n");
      return 0;
```

```
    }
}

static int nrf24l01_release(struct inode *node, struct file *file)
{
   opencount--;
   printk(DEVICE_NAME " released !\n");
   return 0;
}

static struct file_operations nrf24l01_fops = {
   .owner = THIS_MODULE,
   .open = nrf24l01_open,
   .write = nrf24l01_write,
   .read = nrf24l01_read,
   .release = nrf24l01_release,
};

static int __init nrf24l01_init(void)
{
     int ret;

     printk("Initial driver for NRF24L01.....\n");
     ret = register_chrdev(NRF24L01_MAJOR, DEVICE_NAME, &nrf24l01_fops);
     mdelay(10);
     if (ret < 0)
     {
         printk(DEVICE_NAME " can't register major number\n");
         return ret;

     }
     else
     {
```

```
        printk(DEVICE_NAME " register success\n");
 return 0;
    }
}

static void __exit nrf24l01_exit(void)
{
    unregister_chrdev(NRF24L01_MAJOR, DEVICE_NAME);
    printk("NRF24L01 unregister success \n");
}


module_init(nrf24l01_init);
module_exit(nrf24l01_exit);
MODULE_AUTHOR("jammy_lee@163.com");
MODULE_DESCRIPTION("nrf24l01 driver for TQ2440");
MODULE_LICENSE("GPL");
```

测试程序：

```
/*************************************************************/
//文件名：test_ds18b20.c
//功能:测试 linux 下的 ds18b20 程序
//使用说明: (1)
//          (2)
//          (3)
//          (4)
//作者:jammy-lee
//日期:2010-01-18
/*************************************************************/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>
unsigned char TxBuf[32] = {0x00};
int main(void)
{
    int fd = -1;
    int count = 1;

    //fd = open("/dev/nrf24l01", 0);
   fd = open("/dev/nrf24l01", O_RDWR);    //打开 nrf24l01 为可读写文件
    if(fd < 0)
    {
        perror("Can't open /dev/nrf24l01 \n");
        exit(1);
    }
    printf("open /dev/nrf24l01 success \n");
```

```
    while(count <= 5)
    {
        write(fd, &TxBuf , sizeof(TxBuf));
        printf("Sending %d time \n", count);
        usleep(100*1000);

        count++;
    }
    close(fd);
}
```