（本文档为比赛过程中调试无线芯片 NRF24L01 的各种问题及一些分析，后附有相应代码，经过了 IAR4.21A 的测试，在头文件中有相关引脚的定义，移植较为方便。如若有时间，将会整理出更加详细的文档。Coolouba 2010-7-15）

首先，要了解清楚各个寄存器的复位值，还有那些寄存器的设置很重要。

实验中通常用 P0 通道作为实验通道，此时应该将 AUTO_ACK 全部关掉，不是简单屏蔽掉 SPI_RW_Reg(WRITE_REG + EN_AA, 0x01);就可以了，因为 EN_AA 的复位值为全 1，即全部打开自动确认；正确的做法是 SPI_RW_Reg(WRITE_REG + EN_AA, 0x00);在实验中发现使用 P0 通道发送和接收时不能使能自动 ACK，猜想可能是因为 P0 也是自动 ACK 的接收通道造成，有待实验验证。

发送模式的配置和接收模式的配置基本相当，但接收端一定要配置接收有效数据，即执行 SPI_RW_Reg(WRITE_REG + RX_PW_P0,？);问号代表有效数据个数；因为 RX_PW_Px 寄存器的复位值为 0，而 0 代表不合法设置。

P0 通道的接收地址能全部修改，P1 的地址也能全部修改，而其它通道的地址不能全部修改，只能修改最低字节，而高四字节与 P1 相同，在使用其它通道时应当注意。

实验中（P0 通道，无自动确认）接收端收到数据，状态寄存器为 0X40，4 代表 RX_DR 置位，1 到 4 位为 0000，代表 P0 收到数据。当重复接收到数据时，发现只有上电伊始接收端出来的状态为 0x40,而后全为 0x00（当接收到数据后用了写状态寄存器以复位标志位），根据用来指示 P0 通道的 1 到 4 位是 0000 为正确值，怀疑其实是收到数据了的；冲洗了接收寄存器后，每次都能正确显示状态 0x40；

2010-6-9　20:15

实验验证了，在每次接收到数据后将数据读出，不用冲洗数据，状态标志也能正常置位。联想到 FIFO，怀疑有些读取操作被忽略了。

2010-6-9　20:19

翻出了上次比赛用 51 调试时总结的一个东西：

不小心的发现,恰好网上看见有人在问这个。STATUS 和 FIFO_STATUS 两个寄存器中都有 TX_FIFO 标志位，其值与冲洗寄存器和发送状况有关，下面是实验结果。（没有接收端，则 EN_AA 数据没发出）。都 1:22 了  想说什么来着，哦，其实还早嘛。

EN_AA: （发送不成功，若不冲洗，数据保留，否则数据清除）

不 BLUSH_TX　STATUS 中　TX_FULL 1　　，　　FIFO_STATUS 中　TX_FULL　　1；
　　BLUSH_TX　STATUS 中 TX_FULL　0　，　　FIFO_STATUS 中 TX_EMPTY　1；

dis EN_AA: （发送成功，虽然收不到，数据都被清除了）

不 BLUSH_TX　STATUS 中 TX_FULL 0　　，　　FITO_STATUS 中 TX_EMPTY　1；
　　BLUST_TX　STATUS 中　TX_FULL 0　　，　　FIFO_STATUS 中 TX_EMPTY　1；

突然看见读出来的数据有点问题，datasheet 上说读取接收寄存器后，FIFO 自动清除，但实际上好像并不是这样。

发现还是以前那个问题，没有解决，每次发送都要配置发送模式，每次接收也要配置接收模式，按照数据手册的说法，待机下寄存器的值不变，那就范不着每次都配置一下。

2010-6-9　23:55

尝试不在每次发送前都配置所有寄存器，而是拉低 CE 后在置为高，发现还是不行。怀疑是某个寄存器在接收到数据之后复位了，有可能是发送装载数据的寄存器。

2010-6-9　0:08

尝试了重新装载数据，发现仍然不能正常接收。

2010-6-9　0:18

对于网上通用的 IO 模拟 SPI 的驱动程序，装在发射数据不能在 TX_MODE()外执行，虽然那样做后发送端状态寄存器发送位能置位，但接收端不能收到数据，怀疑与 CE 信号有关。

2010-7-11 20:47

实验证明，的确是 CE 信号没有处理所致上述问题。在执行装载数据前置低 CE，装载后置高 CE，能正常发送数据，但是丢包严重。后在置高 CE 后添加延时，问题解决。由此联想到从前调试时出现的发送数据都要重新配置 TX_MODE()，怀疑是其他操作时混乱了 CE 信号。

2010-7-11 20:58

尝试在只配置一次 TX_MODE()，然后每次发射前都重新配置 CE，也不能重复发射，这样否决了上述疑点。

2010-7-11 21:16

问题找到了，就是在于 CE 信号。该例程中 TX_MODE()最后置高 CE，启动发射，所以每次都要执行 TX_MODE()才能发射。TX_MODE()外装载数据后置高 CE 就能发射，而不需要重新配置 TX_MODE()中所有函数。

2010-7-11 21:33

发送与接收使能 AutoACK 组合	发送端状态	接收端状态
使能/不使能	未发送，1F	未接收，0E
使能/使能	发送　2E	接收　40
不使能/使能	发送　2E	未接收　0E

2010-7-11 22:08

NRF24L01SPI.H

```
#ifndef _BYTE_DEF_
#define _BYTE_DEF_
#endif      /* _BYTE_DEF_ */

typedef unsigned char    uchar;
typedef unsigned    int      uint;

/**************          FUNCTION's PROTOTYPES      ****************/


#define PORT          P1OUT
#define PDIR          P1DIR
#define PIN           P1IN

#define BIT(x)  (1 << (x))
#define CE               6                        //P1.3
#define CSN              5
#define CLK              3
#define MOSI             1
#define MISO             2
#define IRQ              4
/*************address width and Pload width******************/

#define  TX_ADR_WIDTH        5                                                    //  5  bytes
TX(RX) address width
#define  TX_PLOAD_WIDTH       5                                                   //  1  bytes
TX payload


//************    SPI(nRF24L01) commands    *********************/

#define READ_REG              0x00    // Define read command to register
#define WRITE_REG             0x20    // Define write command to register
#define RD_RX_PLOAD           0x61    // Define RX payload register address
```

```
#define WR_TX_PLOAD         0xA0    // Define TX payload register address
#define FLUSH_TX            0xE1    // Define flush TX register command
#define FLUSH_RX            0xE2    // Define flush RX register command
#define REUSE_TX_PL         0xE3    // Define reuse TX payload register command
#define NOP                 0xFF    // Define No Operation, might be used to read status register



/*********** SPI(nRF24L01) registers(addresses) ***************/

#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

/***************nrf24l01 fuctions    ******************/

void NRF24L01IO_initial();
uchar SPI_RW(uchar byte);
uchar SPI_RW_Reg(uchar reg, uchar value);
uchar SPI_Read(uchar reg);
uchar SPI_Read_Buf(uchar reg, uchar *pBuf, uchar bytes);
uchar SPI_Write_Buf(uchar reg, uchar *pBuf, uchar bytes);
void RX_Mode(void);
```

```
void TX_Mode(void);
```

NRF24L01.C

```
#include <msp430xxxx.h>
#include "NRF24L01.H"

typedef unsigned char uchar;


uchar rx_buf1[TX_PLOAD_WIDTH];
uchar                                   tx_buf1[TX_PLOAD_WIDTH]={0x33,0x44,0x55,0x66,0x77};
//THE 0 FOR TRANSMISSION

uchar  TX_ADDRESS[TX_ADR_WIDTH]  =  {0x34,0x43,0x10,0x10,0x01};            // Define a static
TX address ;


///////////////端口初试化//////////////////////////
void NRF24L01IO_initial()
{
   PDIR |= BIT(CSN) + BIT(CE) + BIT(CLK) + BIT(MOSI);
   PORT &=~BIT(CE);                                                  //CHIP DISABLE
   PORT  |=  BIT(CSN);                                                 //CSN  IS  PULL  HIGH.DISABLE  THE
OPERATION
   PORT &=~BIT(CLK);                                                  //CLK IS LOW
}




//////////////////////读写操作////////////////////////////////////
uchar SPI_RW(uchar byte)
{
      uchar bit_ctr;
            PORT &=~(BIT(CLK));

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

```
                if(byte & 0x80)
        PORT |= BIT(MOSI);
                else
                {
                PORT &= ~(BIT(MOSI));                   // output 'byte', MSB to MOSI
                }

        byte = (byte << 1);                     // shift next bit into MSB..
        PORT |=BIT(CLK);                        // Set clk high..
                if(PIN&(BIT(MISO)))
        byte |= BIT0;                   // capture current MISO bit
                else
                {
                byte &=~BIT0;
                }
        PORT &=~BIT(CLK);                       // set    clk low
    }
        PORT &=~(BIT(MOSI));                            //PULL DOWN THE MOSI
    return(byte);                               // return read byte
}


//////////////////////读写寄存器/////////////////////////////////////////////
uchar SPI_RW_Reg(uchar reg, uchar value)
{
    uchar status;

    PORT &=~BIT(CSN);                           // CSN low, init SPI transaction
    status = SPI_RW(reg);                       // select register
    SPI_RW(value);                              // ..and write value to it..
    PORT |=BIT(CSN);                            // CSN high again

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


////////////////////////读一个字节从 24L01/////////////////////////////////////
uchar SPI_Read(uchar reg)
{
    uchar reg_val;

    PORT   &=~BIT(CSN);                                 // CSN low, initialize SPI
communication...
```

```
    SPI_RW(reg);                                        // Select register to read from..
    reg_val = SPI_RW(0);                                // ..then read registervalue
    PORT |=BIT(CSN);                                    // CSN high, terminate SPI
communication

    return(reg_val);                                    // return register value
}

//////////////////////read RX payload, Rx/Tx address//////////////////////


uchar SPI_Read_Buf(uchar reg, uchar *pBuf, uchar bytes)
{
    uchar status,byte_ctr;

    PORT &=~BIT(CSN);                                   // Set CSN low, init SPI tranaction
    status = SPI_RW(reg);                               // Select register to write to and read
status byte

    for(byte_ctr=0;byte_ctr<bytes;byte_ctr++)
        pBuf[byte_ctr] = SPI_RW(0);                     // Perform SPI_RW to read
byte from nRF24L01

    PORT |=BIT(CSN);                                    // Set CSN high again

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

//////////////////////write TX payload, Rx/Tx address////////////////////////////

uchar SPI_Write_Buf(uchar reg, uchar *pBuf, uchar bytes)
{
    uchar status,byte_ctr;

    PORT &=~BIT(CSN);                                   // Set CSN low, init SPI
tranaction
    status = SPI_RW(reg);                               // Select register to write to and
read status byte
    for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)         // then write all byte in buffer(*pBuf)
        SPI_RW(*pBuf++);
    PORT |=BIT(CSN);                                    // Set CSN high again
    return(status);                                     // return nRF24L01 status
byte
}
```

```
//////////////////////initializes one nRF24L01 device////////////////////////////////
/********接收模式，包括写接收端地址、配置自动 ACK、接收通道使能、频道选择、接收数据位数*******/
void RX_Mode(void)
{

    PORT &=~BIT(CE);
        SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);                                   // Set PWR_UP bit, enable CRC(2 bytes) & Prim:RX. RX_DR enabled..

    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH); // Use the same address on the RX device as the TX device

    SPI_RW_Reg(WRITE_REG + EN_AA, 0x00);                                    // Enable Auto.Ack:Pipe0
    SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);                                // Enable Pipe0
    SPI_RW_Reg(WRITE_REG + RF_CH, 40);                                      // Select RF channel 40
    SPI_RW_Reg(WRITE_REG + RX_PW_P0, TX_PLOAD_WIDTH);                       // Select same RX payload width as TX Payload width
    SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);                                 // TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR

    PORT |=BIT(CE);                                                         // Set CE pin high to enable RX device

  //  This device is now ready to receive one packet of 16 bytes payload from a TX device sending to address
  //  '3443101001', with auto acknowledgment, retransmit count of 10, RF channel 40 and datarate = 2Mbps.

}

//////////////////////TRANS FUNCTION/////////////////////////////////////////////////////
/*****发送模式，写发送地址、用于自动 ACK 的接收通道地址、配置自动 ACK、接收通道使能（接收 ACK 信号)*******/
void TX_Mode(void)
{

        unsigned int j;
    PORT &=~BIT(CE);
        SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);                                   // Set
```

```
PWR_UP bit, enable CRC(2 bytes) & Prim:TX. MAX_RT & TX_DS enabled..

    SPI_Write_Buf(WRITE_REG + TX_ADDR, TX_ADDRESS, TX_ADR_WIDTH);        // Writes
TX_Address to nRF24L01
    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH); // RX_Addr0
same as TX_Adr for Auto.Ack
    SPI_Write_Buf(WR_TX_PLOAD, tx_buf1, TX_PLOAD_WIDTH);                  // Writes data
to TX payload

    SPI_RW_Reg(WRITE_REG + EN_AA, 0x01);                                    // Enable
Auto.Ack:Pipe0
    SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);                                // Enable
Pipe0
    //SPI_RW_Reg(WRITE_REG + SETUP_RETR, 0x1a);                             // 500us +
86us, 10 retrans...
    SPI_RW_Reg(WRITE_REG + RF_CH, 40);                                     // Select RF
channel 40
    SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);                                      //
TX_PWR:0dBm, Datarate:2Mbps, LNA:HCURR

    //PORT |=BIT(CE);
        for(j=1000;j>0;j--);
}
```