NRF24L01 使用文档

基于 c8051f330 单片机

目　录

芯片简介

NRF24L01 是 NORDIC 公司最近生产的一款无线通信通信芯片，采用 FSK 调制，内部集成 NORDIC 自己的 Enhanced Short Burst 协议。可以实现点对点或是 1 对 6 的无线通信。无线通信速度可以达到 2M（bps）。NORDIC 公司提供通信模块的 GERBER 文件，可以直接加工生产。嵌入式工程师或是单片机爱好者只需要为单片机系统预留 5 个 GPIO，1 个中断输入引脚，就可以很容易实现无线通信的功能，非常适合用来为 MCU 系统构建无线通信功能。

1 NRF24L01 功能框图

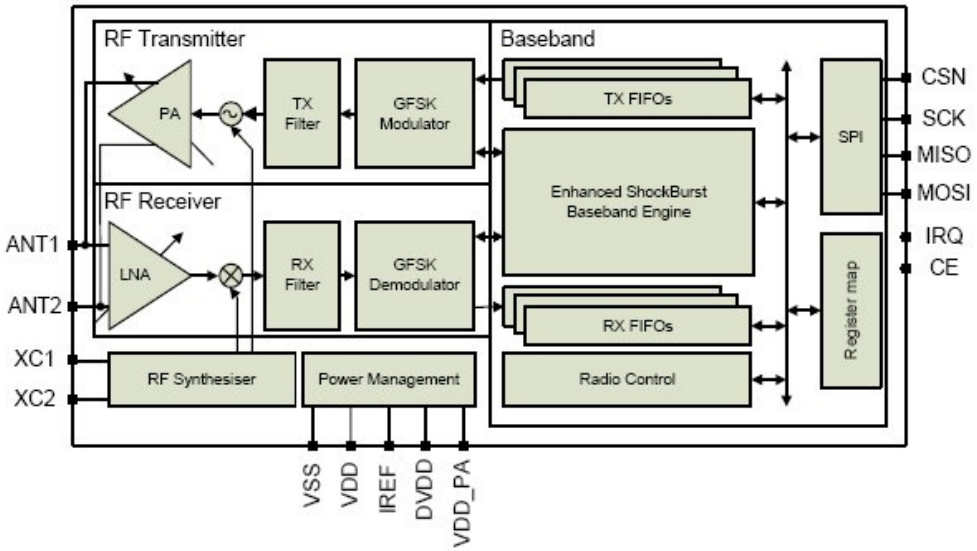

Fig.1 NRF24L01 BLOCK DIAGRAM

NRF24L01 的框图如 Fig.1 所示，从单片机控制的角度来看，我们只需要关注 Fig.1 右面的六个控制和数据信号，分别为 CSN、SCK、MISO、MOSI、IRQ、CE。

CSN：芯片的片选线，CSN 为低电平芯片工作。

SCK：芯片控制的时钟线（SPI 时钟）

MISO：芯片控制数据线（Master input slave output）

MOSI：芯片控制数据线（Master output slave input）

IRQ：中断信号。无线通信过程中 MCU 主要是通过 IRQ 与 NRF24L01 进行通信。

CE： 芯片的模式控制线。 在 CSN 为低的情况下，CE 协同 NRF24L01 的 CONFIG 寄存器共同决定 NRF24L01 的状态（参照 NRF24L01 的状态机）。

2 NRF24L01 状态机

NRF24L01 的状态机见 Fig.2 所示，对于 NRF24L01 的固件编程工作主要是参照 NRF24L01 的状态机。主要有以下几个状态

Power Down Mode：掉电模式

Tx Mode：发射模式

Rx Mode：接收模式

Standby-1Mode：待机 1 模式

Standby-2 Mode：待机 2 模式

上面五种模式之间的相互切换方法以及切换所需要的时间参照 Fig.2。

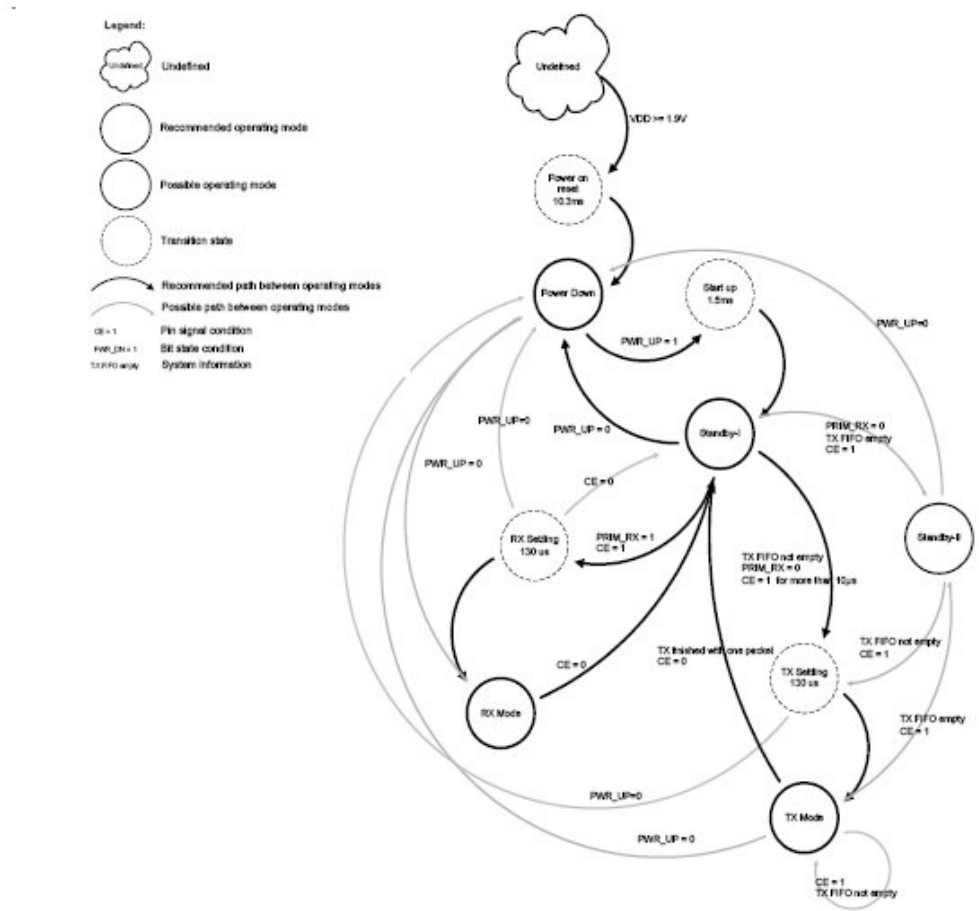

Fig.2 NRF24L01 State Machine

对 24L01 的固件编程的基本思路如下：

1） 置 CSN 为低，使能芯片，配置芯片各个参数。（过程见 3.Tx 与 Rx 的配置过程）配置参数在 Power Down 状态中完成。

2） 如果是 Tx 模式，填充 Tx FIFO。

3） 配置完成以后，通过 CE 与 CONFIG 中的 PWR_UP 与 PRIM_RX 参数确定 24L01 要切换到的状态。

Tx Mode：PWR_UP=1;　PRIM_RX=0;　CE=1 (保持超过 10us 就可以)；

Rx Mode:　PWR_UP=1;　PRIM_RX=1;　CE=1;

4) IRQ 引脚会在以下三种情况变低：

Tx FIFO 发完并且收到 ACK（使能 ACK 情况下）

Rx FIFO 收到数据

达到最大重发次数

将 IRQ 接到外部中断输入引脚，通过中断程序进行处理。

3 Tx 与 Rx 的配置过程

本节只是叙述了采用 ENHANCED SHORT BURST 通信方式的 Tx 与 Rx 的配置及通信过程，熟悉了 24L01 以后可以采用别的通信方式。

3.1 Tx 模式初始化过程

初始化步骤	24L01 相关寄存器
1）写 Tx 节点的地址	**TX_ADDR**
2）写 Rx 节点的地址（主要是为了使能 Auto Ack）	**RX_ADDR_P0**
3）使能 AUTO ACK	**EN_AA**
4）使能 PIPE 0	**EN_RXADDR**
5）配置自动重发次数	**SETUP_RETR**
6）选择通信频率	**RF_CH**
7）配置发射参数（低噪放大器增益、发射功率、无线速率）	**RF_SETUP**
8) 选择通道 0 有效数据宽度	**Rx_Pw_P0**
9）配置 24L01 的基本参数以及切换工作模式	**CONFIG**

Tx 模式初始化过程

1）写 Tx 节点的地址 TX_ADDR

2）写 Rx 节点的地址（主要是为了使能 Auto Ack） RX_ADDR_P0

3）使能 AUTO ACK EN_AA

4）使能 PIPE 0 EN_RXADDR

5）配置自动重发次数 SETUP_RETR

6）选择通信频率 RF_CH

7）配置发射参数（低噪放大器增益、发射功率、无线速率） RF_SETUP

8) 选择通道 0 有效数据宽度 Rx_Pw_P0

9）配置 24L01 的基本参数以及切换工作模式 CONFIG。

3.2 Rx 模式初始化过程

初始化步骤	24L01 相关寄存器
1）写 Rx 节点的地址	**RX_ADDR_P0**
2）使能 AUTO ACK	**EN_AA**
3）使能 PIPE 0	**EN_RXADDR**
4）选择通信频率	**RF_CH**
5) 选择通道 0 有效数据宽度	**Rx_Pw_P0**
6）配置发射参数（低噪放大器增益、发射功率、无线速率）	**RF_SETUP**
7）配置 24L01 的基本参数以及切换工作模式	**CONFIG**

Rx 模式初始化过程：

初始化步骤　24L01 相关寄存器

1）写 Rx 节点的地址　RX_ADDR_P0

2）使能 AUTO ACK　 EN_AA

3）使能 PIPE 0　 EN_RXADDR

4）选择通信频率 RF_CH

5) 选择通道 0 有效数据宽度　 Rx_Pw_P0

6）配置发射参数（低噪放大器增益、发射功率、无线速率）　 RF_SETUP

7）配置 24L01 的基本参数以及切换工作模式 CONFIG。

4 控制程序详解

4.1 函数介绍

NRF24L01 的控制程序主要包括以下几个函数

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);

4.1.1 uchar SPI_RW(uchar byte)

```
uchar SPI_RW(uchar byte)
{
    uchar bit_ctr;
    for(bit_ctr=0;bit_ctr<8;bit_ctr++)     // output 8-bit
    {
        MOSI = (byte & 0x80);              // output 'byte', MSB to MOSI
        byte = (byte << 1);                // shift next bit into MSB..
        SCK = 1;                           // Set SCK high..
        byte |= MISO;                      // capture current MISO bit
        SCK = 0;                           // ..then set SCK low again
    }
    return(byte);                          // return read byte
}
```

最基本的函数，完成 GPIO 模拟 SPI 的功能。将输出字节（MOSI）从 MSB 循环输出，同时将输入字节（MISO）从 LSB 循环移入。上升沿读入，下降沿输出。（从 SCK 被初始化为低电平可以判断出）。

4.1.2 uchar SPI_RW_Reg (uchar reg, uchar value)

```
uchar SPI_RW_Reg(uchar reg, uchar value)
{
    uchar status;
    CSN = 0;                        // CSN low, init SPI transaction
    status = SPI_RW(reg);          // select register
    SPI_RW(value);                 // ..and write value to it..
    CSN = 1;                        // CSN high again

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

寄存器访问函数：用来设置 24L01 的寄存器的值。基本思路就是通过 WRITE_REG 命令（也就是 0x20+寄存器地址）把要设定的值写到相应的寄存器地址里面去，并读取返回值。对于函数来说也就是把 value 值写到 reg 寄存器中。

需要注意的是，访问 NRF24L01 之前首先要 enable 芯片（CSN=0；），访问完了以后再 disable 芯片（CSN=1；）。

4.1.3 uchar SPI_Read (uchar reg);

```
uchar SPI_Read(uchar reg)
{
    uchar reg_val;

    CSN = 0;                        // CSN low, initialize SPI communication...
```

```
    SPI_RW(reg);                    // Select register to read from..
    reg_val = SPI_RW(0);        // ..then read registervalue
    CSN = 1;                        // CSN high, terminate SPI communication

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

读取寄存器值的函数：基本思路就是通过 READ_REG 命令（也就是 0x00+寄存器地址），把寄存器中的值读出来。对于函数来说也就是把 reg 寄存器的值读到 reg_val 中去。

4.1.4 uchar SPI_Read_Buf (uchar reg, uchar *pBuf, uchar bytes);

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

    CSN = 0;                                // 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

    CSN = 1;                                    // Set CSN high again

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

接收缓冲区访问函数：主要用来在接收时读取 FIFO 缓冲区中的值。基本思路就是通过 READ_REG 命令把数据从接收 FIFO（RD_RX_PLOAD）中读出并存到数组里面去。

4.1.5 uchar SPI_Write_Buf (uchar reg, uchar *pBuf, uchar bytes);

```
uchar SPI_Write_Buf(uchar reg, uchar *pBuf, uchar bytes)
```

```
{
    uchar status,byte_ctr;

    CSN = 0;                    // Set CSN low, init SPI tranaction
    status = SPI_RW(reg);    // Select register to write to and read status byte
    Uart_Delay(10);
    for(byte_ctr=0;byte_ctr<bytes;byte_ctr++) // then write all byte in buffer(*pBuf)
    SPI_RW(*pBuf++);
    CSN = 1;                  // Set CSN high again
    return(status);           // return nRF24L01 status byte
}
```

发射缓冲区访问函数：主要用来把数组里的数放到发射 FIFO 缓冲区中。基本思路就是通过 WRITE_REG 命令把数据存到发射 FIFO（WR_TX_PLOAD）中去。

4.1.6 void RX_Mode(void)

设定 24L01 为接收方式，配置过程详见 3.2 Rx 模式初始化过程。

```
void RX_Mode(void)
{
    CE=0;
    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH);
    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 + RF_CH, 40);        // Select RF channel 40
    SPI_RW_Reg(WRITE_REG + RX_PW_P0, TX_PLOAD_WIDTH);
    SPI_RW_Reg(WRITE_REG + RF_SETUP, 0x07);
    SPI_RW_Reg(WRITE_REG + CONFIG, 0x0f);     // Set PWR_UP bit, enable CRC(2 bytes)
    & Prim:RX. RX_DR enabled..

    CE = 1; // 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.

}
```

4.1.7 void TX_Mode(void)

设定 24L01 为发送方式，配置过程详见 3.1 Tx 模式初始化过程。

```
void TX_Mode(void)
{
    CE=0;

    SPI_Write_Buf(WRITE_REG + TX_ADDR, TX_ADDRESS, TX_ADR_WIDTH);
    SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH);
    SPI_Write_Buf(WR_TX_PLOAD, tx_buf, 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
    SPI_RW_Reg(WRITE_REG + CONFIG, 0x0e);     // Set PWR_UP bit, enable CRC(2 bytes)
    & Prim:TX. MAX_RT & TX_DS enabled..
    CE=1;

}
```

4.2 NRF24L01 相关命令的宏定义

nRF24L01 的基本思路就是通过固定的时序与命令，控制芯片进行发射与接收。控制命令如 FIG..4.2.1 所示。

SPI 接口指令		
指令名称	指令格式	操作
R_REGISTER	000A AAAA	读配置寄存器。AAAAA 指出读操作的寄存器地址
W_REGISTER	001A AAAA	写配置寄存器。AAAAA 指出写操作的寄存器地址 只能在掉电模式或待机模式下操作。
R_RX_PAYLOAD	0110 0001	读 RX 有效数据：1-32 字节。读操作全部从字节 0 开始。 当读 RX 有效数据完成后，FIFO 寄存器中有效数据被清除。 应用于接收模式下。
W_RX_PAYLOAD	1010 0000	写 TX 有效数据：1-32 字节。写操作从字节 0 开始。 应用于发射模式下
FLUSH_TX	1110 0001	清除 TX FIFO 寄存器，应用于发射模式下。
FLUSH_RX	1110 0010	清除 RX FIFO 寄存器，应用于接收模式下。 在传输应答信号过程中不应执行此指令。也就是说，若传输应答信号过程中执行此指令的话将使得应答信号不能被完整的传输。
REUSE_TX_PL	1110 0011	应用于发射端 重新使用上一包发射的有效数据。当 CE=1 时，数据被不断重新发射。 在发射数据包过程中必须禁止数据包重利用功能。
NOP	1111 1111	空操作。可用来读状态寄存器。

FIG.4.2.1

前面提到的函数也要与这些命令配合使用,比如

```
SPI_RW_Reg(WRITE_REG + EN_RXADDR, 0x01);

SPI_Write_Buf(WRITE_REG + TX_ADDR, TX_ADDRESS, TX_ADR_WIDTH);
```

相关命令的宏定义如下：

```
#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
```

4.3 NRF24L01 相关寄存器地址的宏定义

```
#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
```

5 实际通信过程示波器图

对于 NRF24L01 的编程主要是通过命令（WRITE_REG，READ_REG 等等），控制线 CE、CSN）以及中断信号 IRQ 共同完成的。

对于发射节点，如果使能 ACK 与 IRQ 功能，则当通信成功以后（也就是发射节点收到了接收节点送回的 ACK 信号）IRQ 线会置低。

对于接收节点，如果使能ACK与IRQ功能，则当通信成功以后（主要是根据Enhanced ShockBurst协议认为成功收到了有效数据宽度的数据）IRQ线会置低。

根据以上两种情况，用示波器抓了以下几个图形，分别介绍如下：

1）发射节点 CE 与 IRQ 信号

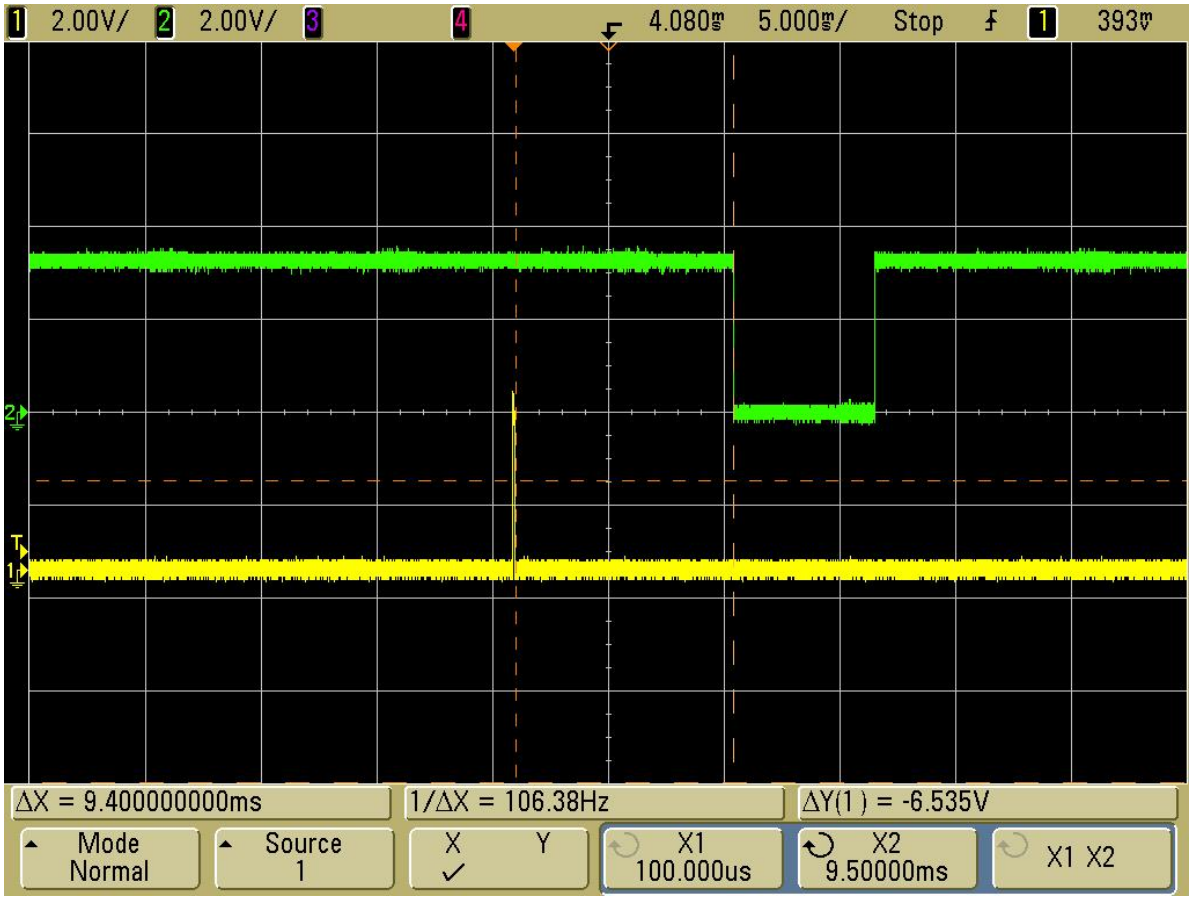

FIG5.1

黄色信号是 CE，绿色信号是 IRQ，当把节点配置为发射节点以后，将要传送的数据通过 SPI_Write_Buf(WRITE_REG + RX_ADDR_P0, TX_ADDRESS, TX_ADR_WIDTH)函数送到发送 FIFO 缓冲区。CE 为高超过 10us，缓冲区中的数据通过无线向外发出。如果使能 IQR 的全部功能（TX_DS,RX_DS,MAX_RT）当发送节点收到接收节点发来的 ACK（表示接收节点成功收到信号）或是达到最大发射次数，IRQ 会变为低电平，同时 CONFIG 的相关标志位（）会置 1。清除标志位（向 CONFIG 的标志位写 1）以后，IRQ 又变为高电平。

从 FIG5.1 可以看出，CE 置高后将近 10msIRQ 才置低。IRQ 置低是由于达到最大发射次数（MAX_RT=1）,出现该情况可能是由于接收节点的配置与发射节点不符（例如发射接收频率不同，或者发射接收字节不等），或者根本就没有接收节点（例如接收节点就根本没上电）。

2）SCK 与 IRQ 信号（发送成功）

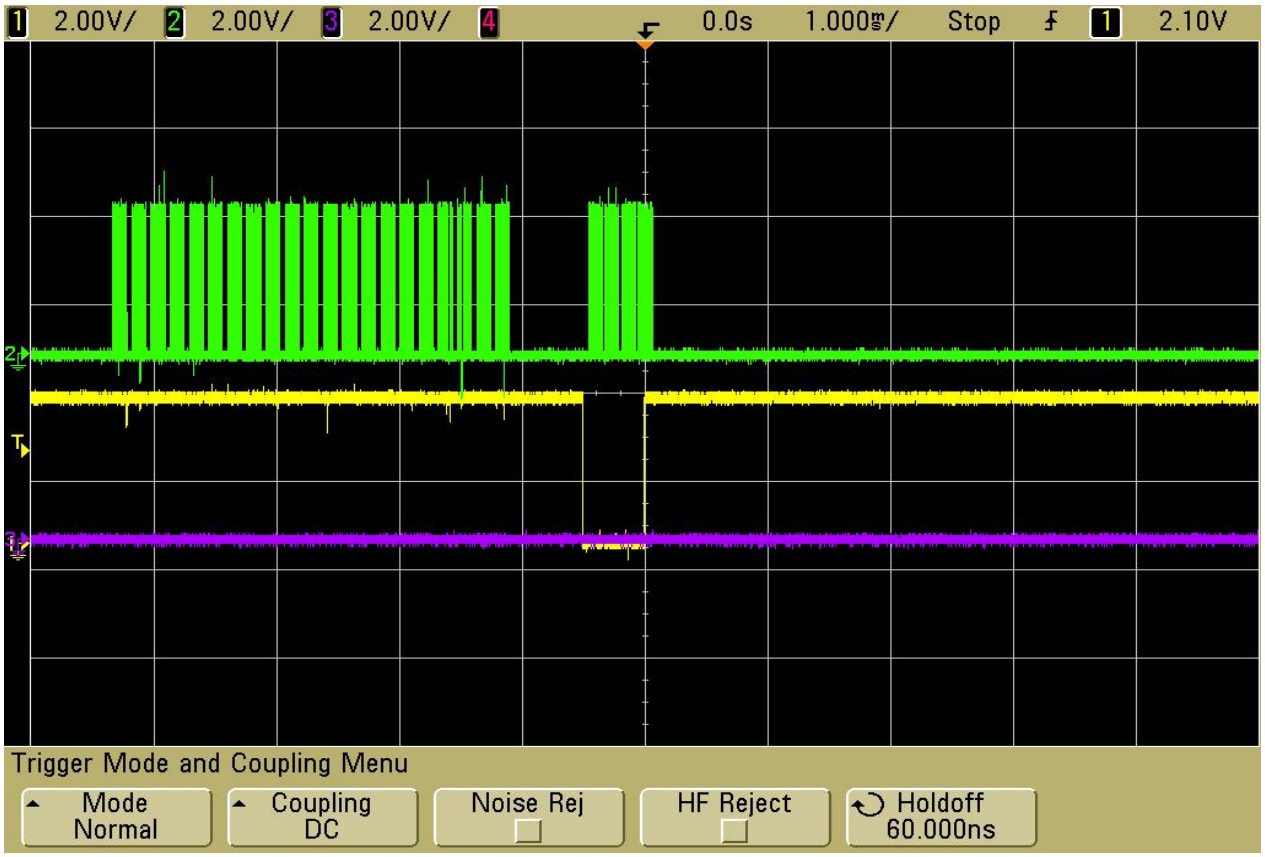

Fig5.2

Fig5.2 中绿色信号是 SCK，黄色信号是 IRQ。第一批绿色信号表示节点的配置过程。MOSI 信号（Fig5.2 中未显示出）在 SCK 的下降延送入 24L01 节点。（配置一个寄存器需要两组 SCK 信号，填充 N 字节的 BUFFER 需要 N+1 组 SCK 信号）。

配置完信号以后，将 CE（Fig5.2 中未显出）置高，则 24L01 开始发送（或接收）数据，当发送（或接收）完成以后（或是达到最大发射次数），IRQ 置低。单片机根据当时的状态进行相应的处理。

第二批绿色信号表示单片机在 IRQ 为低时对 24L01 的处理过程。可以是读 FIFO（作为接收节点时），写 FIFO（作为发射节点时），或是 Reset 24L01（达到最大发射次数时）。

从 Fig5.2 可以看到，从第一批 SCK 的最后一个信号到 IRQ 置低大概需要 1ms

（对比于 Fig5.1 的 12ms），说明通信成功（说明 IRQ 不是 MAX_RT 引起的）。

3）SCK 与 IRQ 信号（发送不成功）

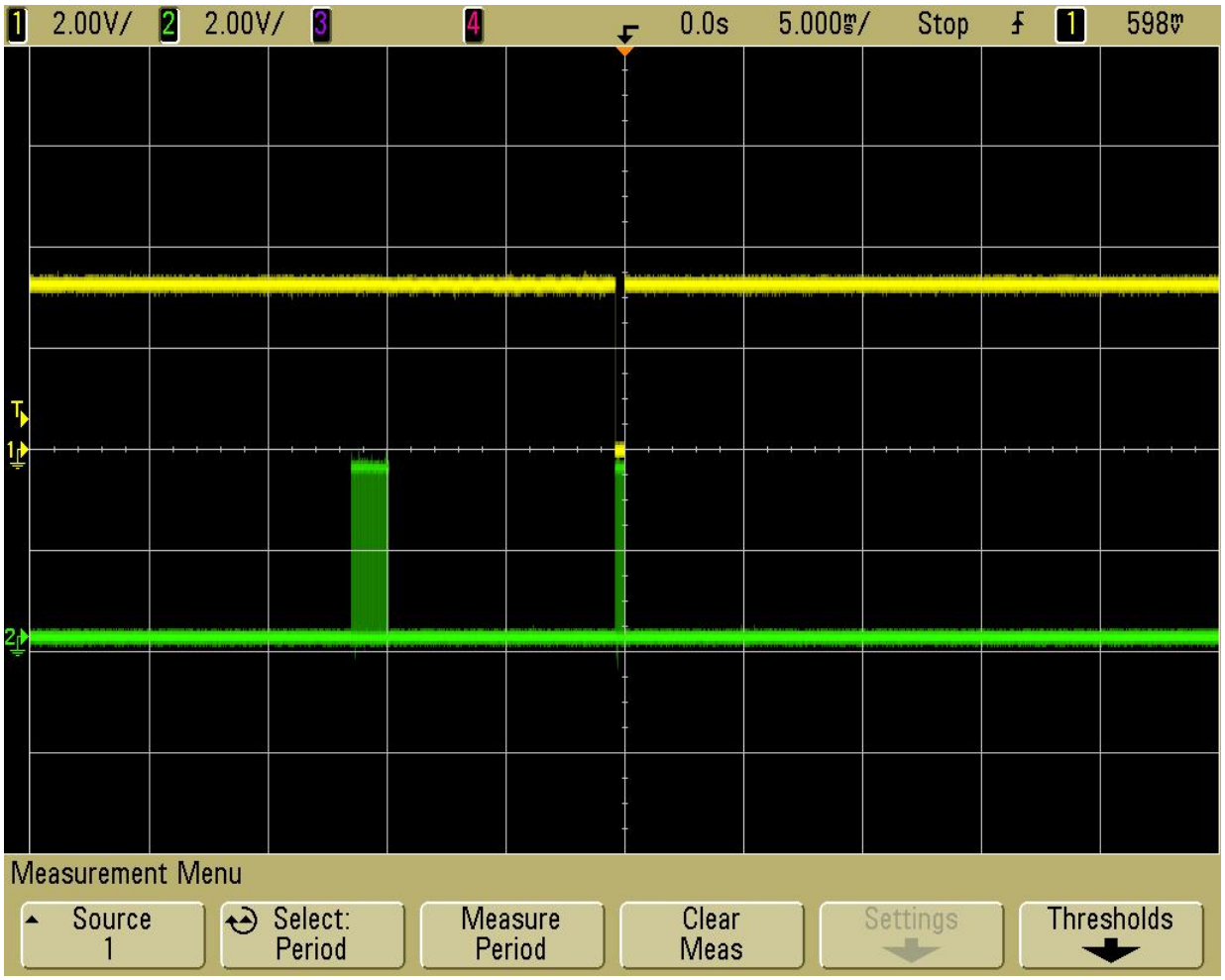

Fig5.3

Fig5.3 与 Fig5.2 类似，只不过从第一批最后一个 SCK 信号到 IRQ 置低的时间间隔变为将近 10ms，表明通信部不成功，IRQ 是由于达到最大发射次数引起的。

4）SCK、IRQ、CE 信号

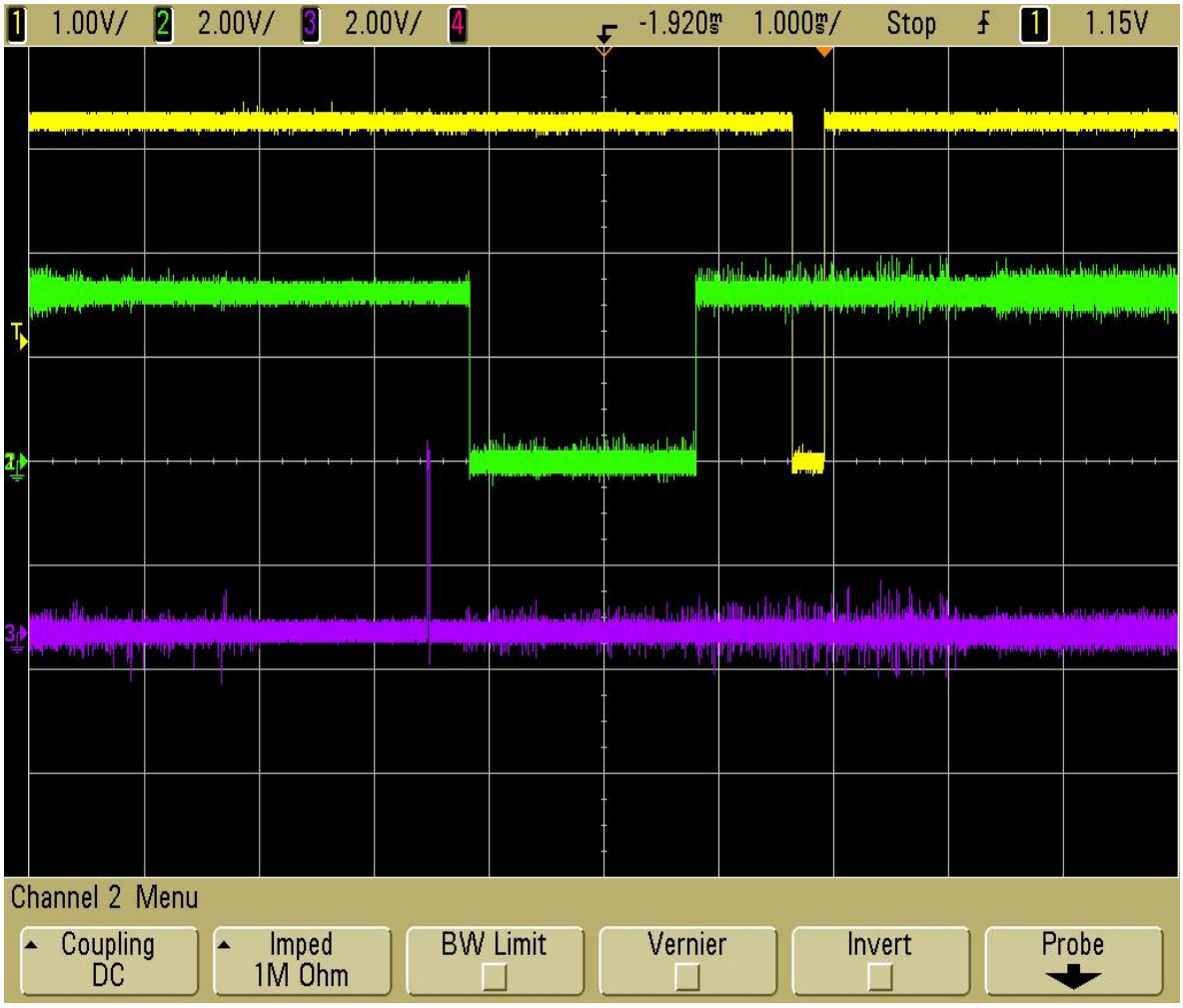

Fig5.4

Fig5.4 中紫色信号是发射端 CE，绿色信号是接收端 IRQ，黄色信号是发射端 IRQ。Fig5.4 表示如下逻辑：

发射节点在配置完成以后（配制过程 Fig5.2 未显示），CE 置高，发射节点 FIFO 中的数据发出；接收节点成功接收到数据，IRQ 置低（从紫色信号与绿色信号之间的时间间隔可以判断出通信成功）；接收节点自动发射 ACK（在发射和接收节点都使能 ACK），发射节点收到 ACK 后 IRQ 置低，表示发送成功。

不同通信环境可能造成发射节点的 IRQ 与接收节点的 IRQ 产生将对的相位变化（表现在示波器上面就是黄色信号靠近绿色信号或者远离绿色信号）。出现这种情况主要是由于不同的通信环境造成接收端发送的 ACK 信号要重发几次才

能被发送端收到。