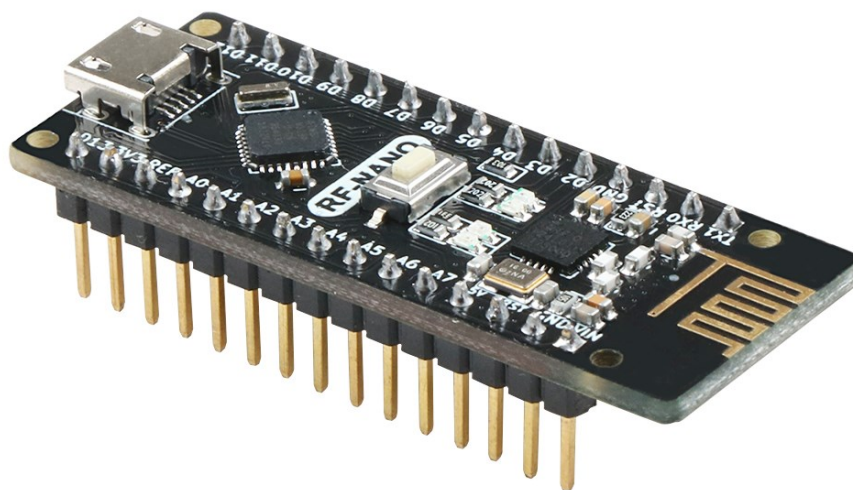


RF-NANO 使用说明书

V.1.2



版本修订历史

Date	Version	Description	Author
2019-7-25	V. 1. 0	创建文档	Abbott. Chen
2019-10-25	V. 1. 1	修改点对点通讯程序	Abbott. Chen
2020-08-12	V. 1. 2	修改 RF-Nano 驱动安装	Carl. Du

目录

第一章 RF-NANO 介绍.....	4
1.1 RF-NANO 概述.....	4
1.2 RF-NANO 处理器介绍.....	6
1.2.1 电源.....	7
1.2.2 存储器.....	7
1.2.3 输入输出.....	7
1.2.4 通信接口.....	7
1.2.5 ATmega328 和 NRF24L01+的通讯方式.....	7
1.2.6 下载程序.....	8
1.2.7 注意要点.....	9
1.3 RF-NANO 驱动安装.....	9
1.3.1 安装 IDE.....	9
1.3.2 安装驱动.....	11
第二章 RF-NANO 工作原理.....	15
2.1 RF-NANO 工作原理介绍.....	15
2.2 配置字.....	16
2.3 RF-NANO 的工作模式.....	17
第三章 实现 RF-NANO 之间通讯.....	19
3.1 实现两个 RF-NANO 点对点通讯.....	19
3.1.1 连线方式.....	19
3.1.2 程序原理.....	20
3.1.3 程序代码.....	20
3.2 实现多个发送对一个接收通讯.....	23
3.2.1 实验原理框图.....	23
3.2.2 程序代码.....	24
3.3 实现一个发送对多个接收通讯.....	28
3.3.1 实验原理框图.....	28
3.3.2 程序代码.....	29
第四章 设置 RF-NANO 发射功率和数据发送速率.....	34
4.1 设置 RF-NANO 发射功率.....	34
4.2 设置 RF-NANO 数据传输速率.....	34

1.1 RF-NANO 概述

RF-NANO 的板子上集成了 NRF24L01+芯片，使得其具有无限收发功能，相当于将一个普通 Nano 板和一个 NRF24L01 模块合二为一，如图 1-1-1，使用起来更加方便，尺寸小。RF-NANO 与常见的 Nano 板的引脚完全一样，方便移植。

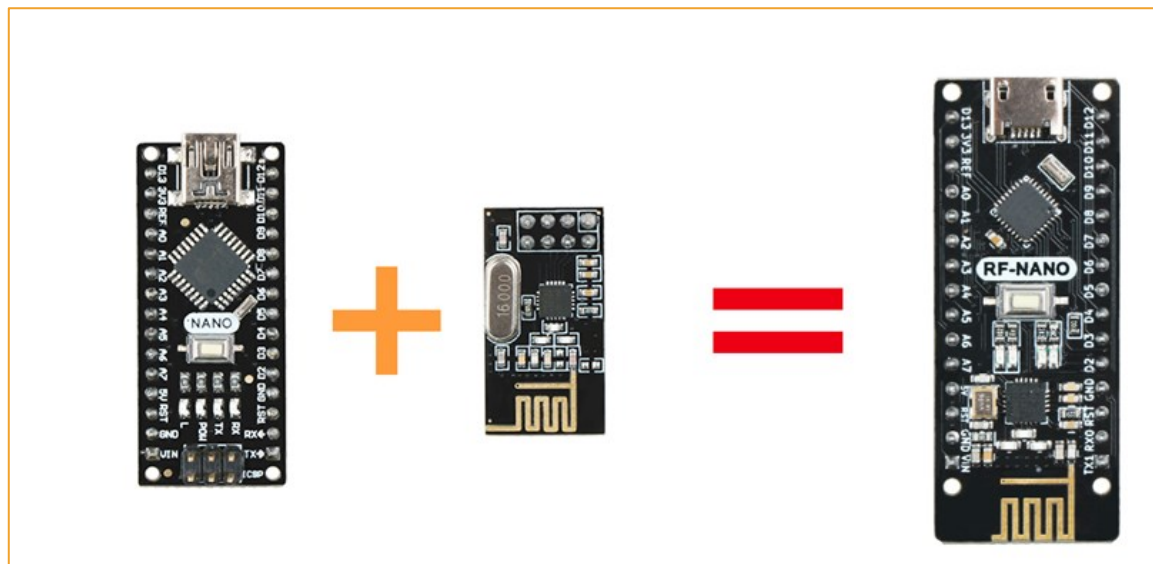


图 1-1-1

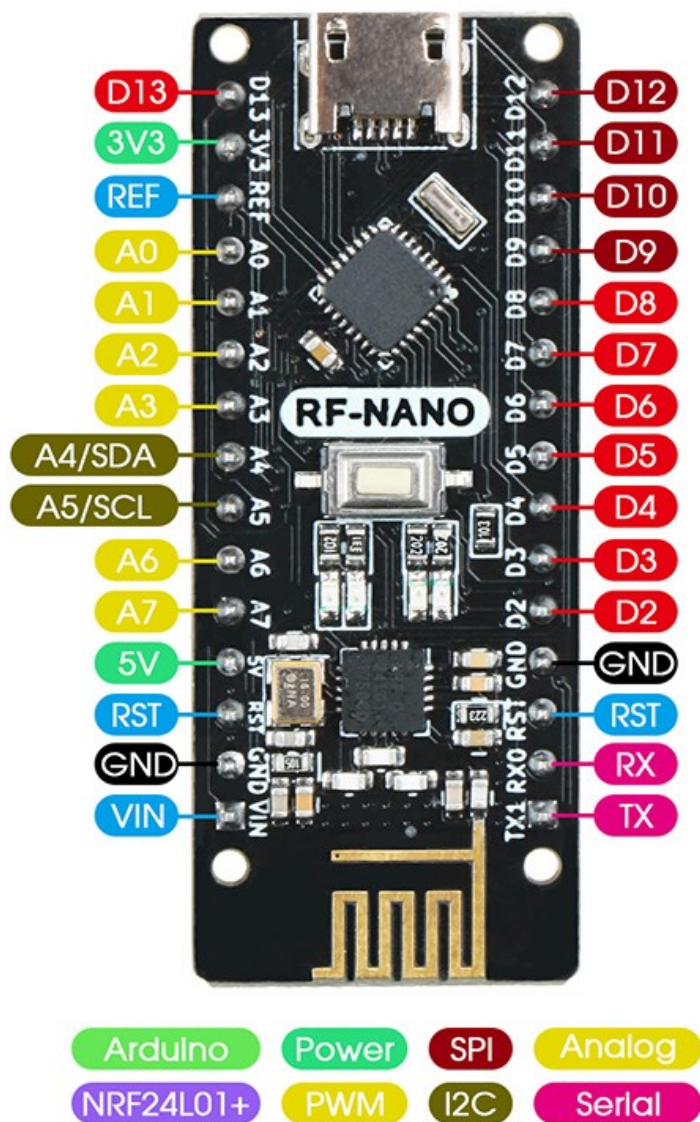


图 1-1-2

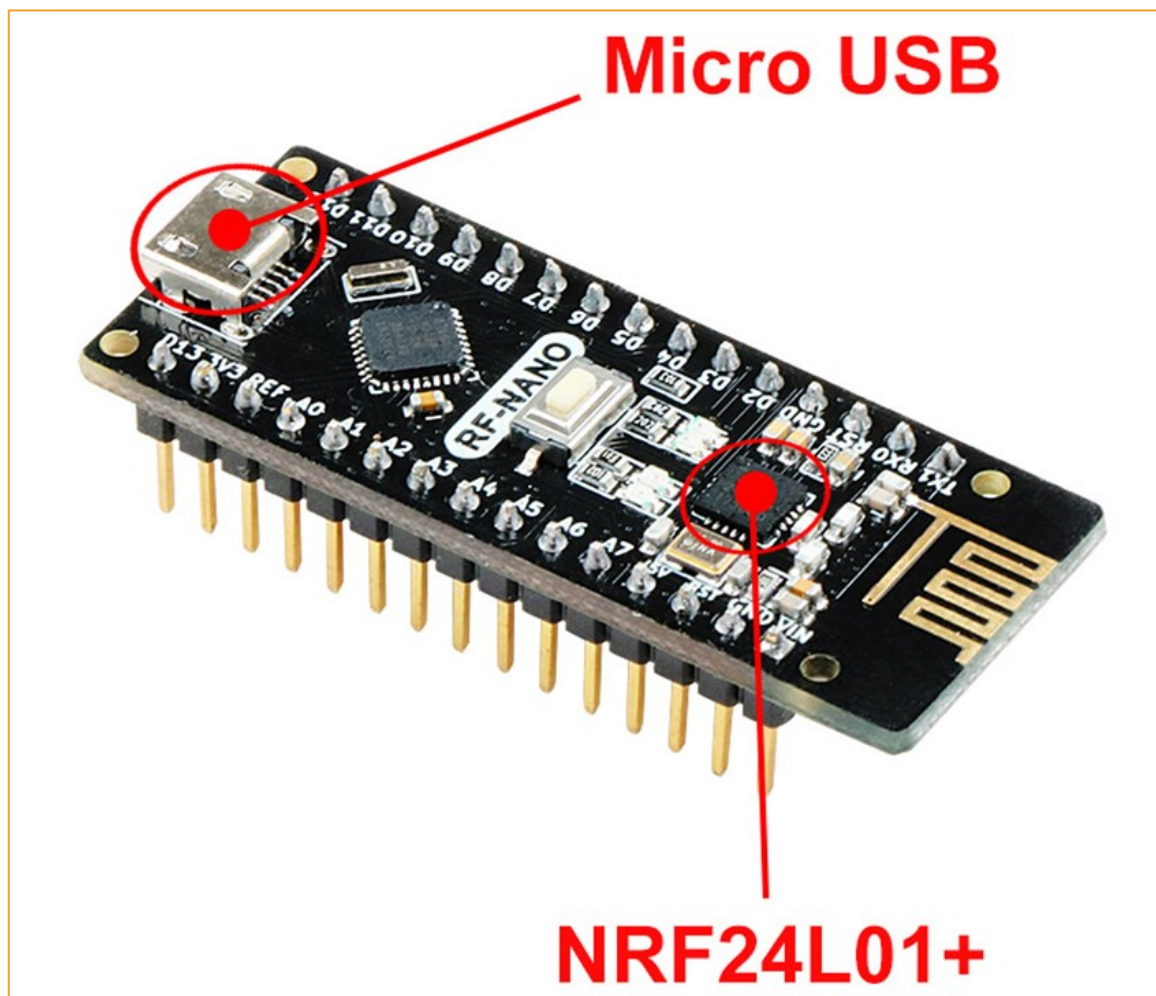


图 1-1-3

1.2 RF-NANO 处理器介绍

Arduino RF-NANO 微处理器是 ATmega328(Nano3.0)，带 USB-Micro 接口，同时具有 14 路数字输入/输出口(其中 6 路可作为 PWM 输出)，8 路模拟输入，一个 16MHz 晶体振荡器，一个 USB-Micro 口，一个 ICSP header 和一个复位按钮。

- ◆ 处理器： ATmega328
- ◆ 工作电压： 5V
- ◆ 输入电压（推荐）： 7-12V
- ◆ 输入电压（范围）： 6-20V
- ◆ 数字 IO 脚： 14 (其中 6 路作为 PWM 输出) (D0~D13)
- ◆ 模拟输入脚： 6 个 (A0~A5)
- ◆ IO 脚直流电流： 40mA
- ◆ Flash Memory : 32KB (其中 2KB 用于 bootloader)
- ◆ SRAM : 2KB

- ◆ EEPROM : 1KB (ATmega328)
- ◆ USB 转串口芯片: CH340
- ◆ 工作时钟: 16 MHZ

1.2.1 电源

Arduino RF-Nano 供电方式: Micro - USB 接口供电和外部 vin 接 7~12V 外部直流电源

1.2.2 存储器

ATmega328 包括了片上 32KB Flash, 其中 2KB 用于 Bootloader。同时还有 2KB SRAM 和 1KB EEPROM。

1.2.3 输入输出

- ◆ 14 路数字输入输出: 工作电压为 5V, 每一路能输出和接入最大电流为 40mA。每一路配置了 20-50K 欧姆内部上拉电阻 (默认不连接)。除此之外, 有些引脚有特定的功能。
- ◆ 串口信号 RX (0 号)、TX (1 号): 提供 TTL 电压水平的串口接收信号, 与 FT232RL 的相应引脚相连。
- ◆ 外部中断 (2 号和 3 号): 触发中断引脚, 可设成上升沿、下降沿或同时触发。
- ◆ 脉冲宽度调制 PWM (3、5、6、9、10、11): 提供 6 路 8 位 PWM 输出。
- ◆ SPI (10(SS), 11(MOSI), 12(MISO), 13(SCK)): SPI 通信接口。
- ◆ LED (13 号): Arduino 专门用于测试 LED 的保留接口, 输出为高时点亮 LED, 反之输出为低时 LED 熄灭。
- ◆ 6 路模拟输入 A0 到 A5: 每一路具有 10 位的分辨率 (即输入有 1024 个不同值), 默认输入信号范围为 0 到 5V, 可以通过 AREF 调整输入上限。除此之外, 有些引脚有特定功能。
- ◆ TWI 接口 (SDA A4 和 SCL A5): 支持通信接口 (兼容 I2C 总线)。
- ◆ AREF: 模拟输入信号的参考电压。
- ◆ Reset: 信号为低时复位单片机芯片。

1.2.4 通信接口

串口: ATmega328 内置的 UART 可以通过数字口 0 (RX) 和 1 (TX) 与外部实现串口通信。

1.2.5 ATmega328 和 NRF24L01+的通讯方式

ATmega328 和 NRF24L01+是 SPI 通讯, 原理图如图 1-2-1 所示;

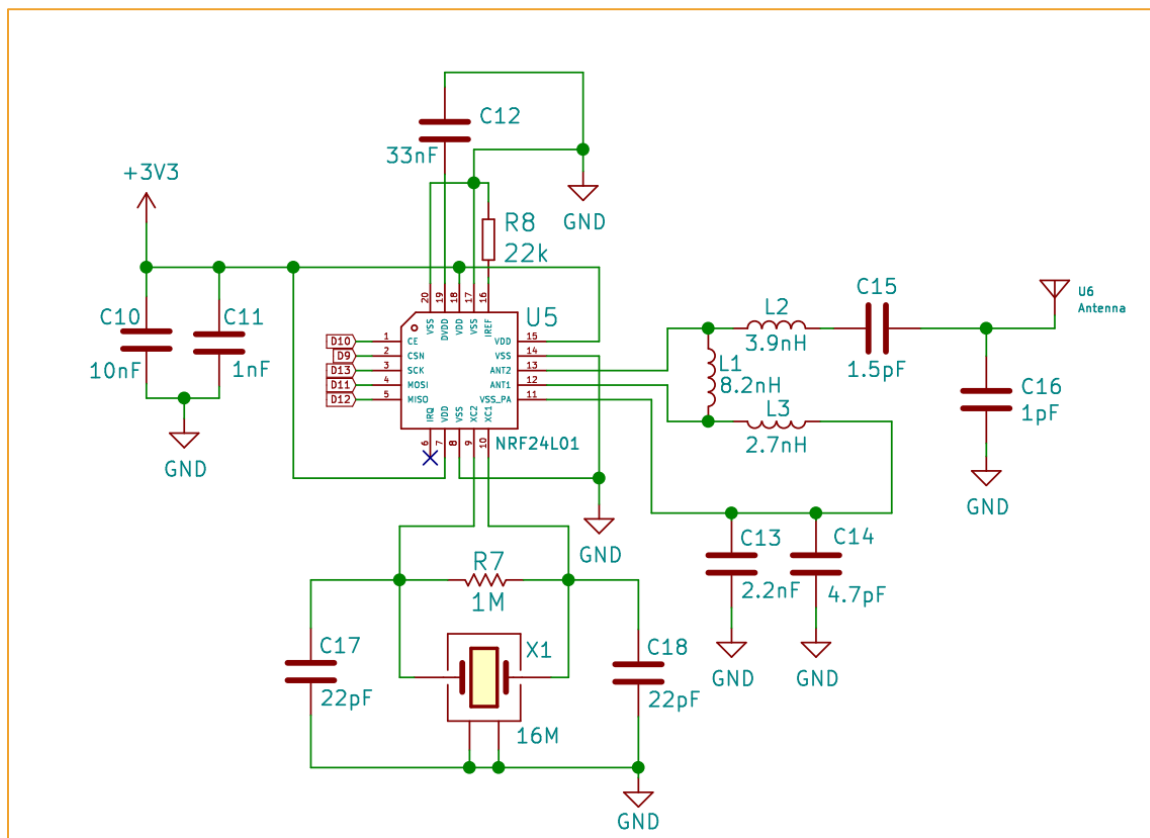


图 1-2-1

ATmega328 和 NRF24L01+芯片引脚连接:

ATmega328 芯片	NRF24L01+芯片
+3.3V	VCC
GND	GND
D9	CSN
D10	CE
D11	MOSI
D12	MISO
D13	SCK

注意: ATmega328 已经被占用的 D9, D10, D11, D12, D13 引脚不能再被复用。

1.2.6 下载程序

Arduino RF-Nano 上的 MCU 已经预置了 bootloader 程序, 因此可以通过 Arduino IDE 软件直接下载程序。也可以直接通过 RF-Nano 上 ICSP header 直接下载程序到 MCU。

1.2.7 注意要点

Arduino RF-Nano 提供了自动复位设计，可以通过主机复位。这样通过 Arduino 软件下在程序到 RF-Nano 中软件可以自动复位，不需要在复位按钮。

1.3 RF-NANO 驱动安装

1.3.1 安装 IDE

本教程需要软件需要 ArduinoIDE 平台，下载地址为：

<https://www.arduino.cc/en/Main/OldSoftwareReleases#previous>，在浏览器打开该连接后，我们可以看到如图 1-3-1 界面，在该界面中，我们可以看到 IDE 的不同版本和不同运行环境，大家根据自己的电脑系统进行下载即可，当然在我们配套的光盘中会有下载好的安装包，但只有 Windows 版本，因为本套教程全都是在 Windows 系统下运行的。

1.6.8	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.7	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.6	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.5	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.4	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.3	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.2	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.1	Windows Windows Installer	MAC OS X MAC OS X Java 7+	Linux 32 Bit Linux 64 Bit	Source code on Github
1.6.0	Windows Windows Installer	MAC OS X MAC OS X Java 7	Linux 32 Bit Linux 64 Bit	Source code on Github
1.5.8 BETA	Windows Windows Installer	MAC OS X MAC OS X Java 7	Linux 32 Bit Linux 64 Bit	Source code on Github

图 1-3-1 ArduinoIDE 下载界面

下载结束后，我们会得到如图 1-3-2 的压缩包，将压缩包进行解压后如图 1-3-3 中的文件，其中“drivers”是驱动软件，在安装“arduino.exe”时会自动安装驱动。因为“arduino.exe”的安装很简单，这里不再讲解，**建议在安装过程中退出杀毒软件**，否则可能会影响 IDE 的安装。安装结束后，再次点击“arduino.exe”即可进入 IDE 程序编写界面。

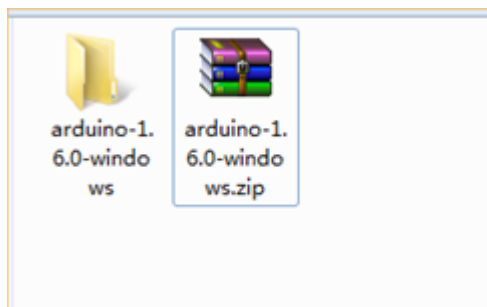


图 1-3-2 ArduinoIDE 安装包

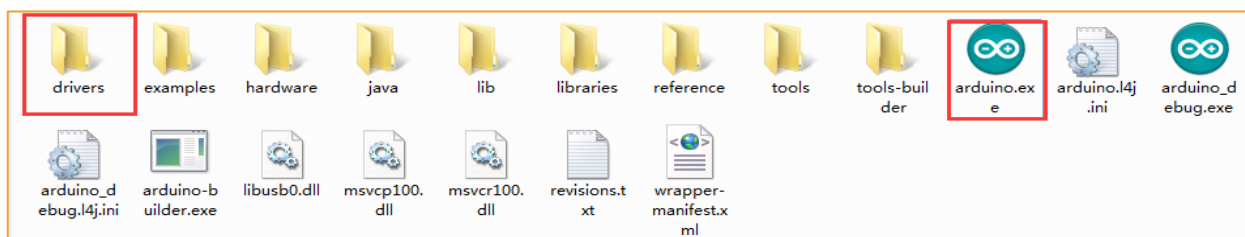


图 1-3-3 解压后的文件

IDE 安装结束后,我们接上 Arduino 主板,右键点击“我的电脑”→“属性”→“设备管理器”→查看“端口 (COM 和 LPT)”, 如果看到如图 1-3-4 所示。

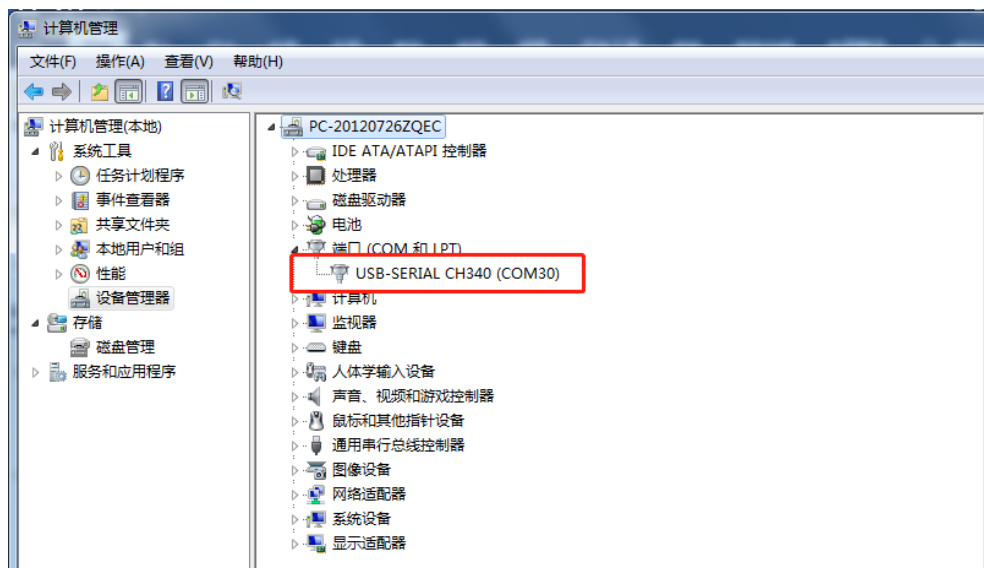


图 1-3-4 驱动安装成功界面

则说明驱动已安装成功,这时我们打开 IDE,在工具栏中选择对应的开发板型号和端口就正常使用了。如果出现如图 1-3-5,则说明电脑没有识别到开发板,需要自己安装驱动程序。

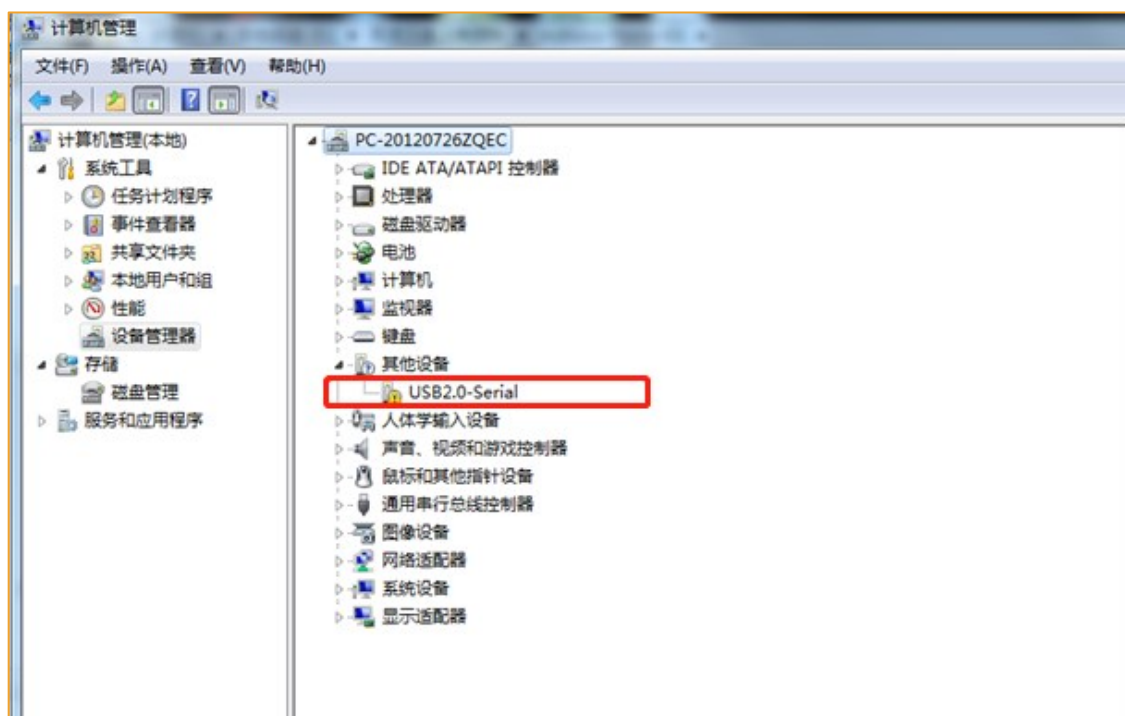


图 1-3-5 驱动未成功安装界面

1.3.2 安装驱动

Windows7 系统驱动安装步骤:

1) 右键点击“我的电脑”→打开设备管理器→查看端口（COM 和 LPT）。此时你会看到一个“USB 串行端口”，右键单击“USB 串行端口”并选择“更新驱动程序软件”选项。

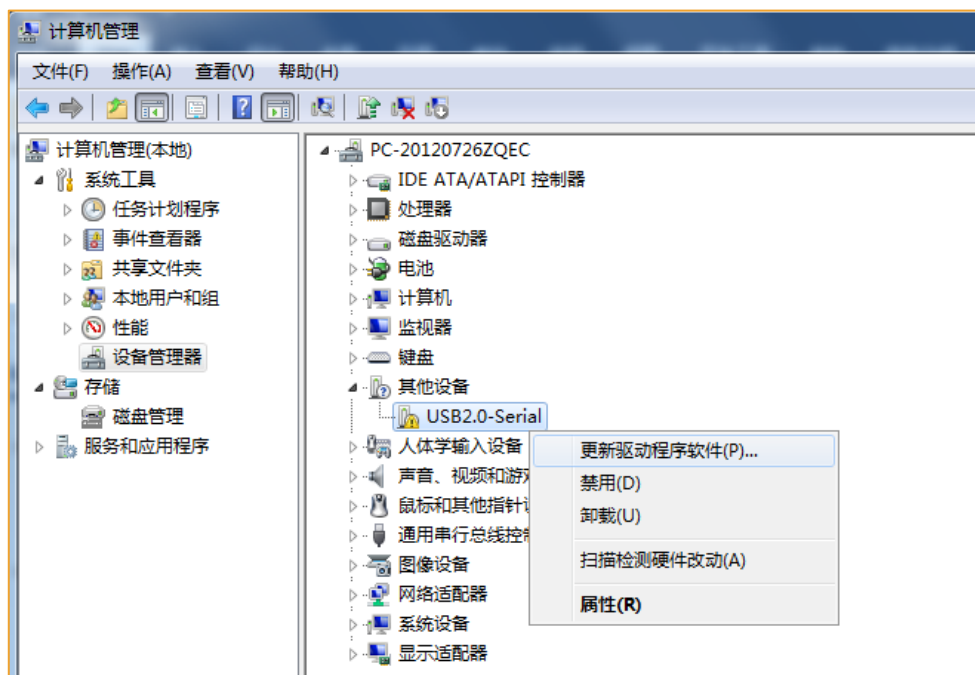


图 1-3-6 更新驱动界面

2) 接下来, 选择“浏览计算机以查找驱动程序软件”选项。

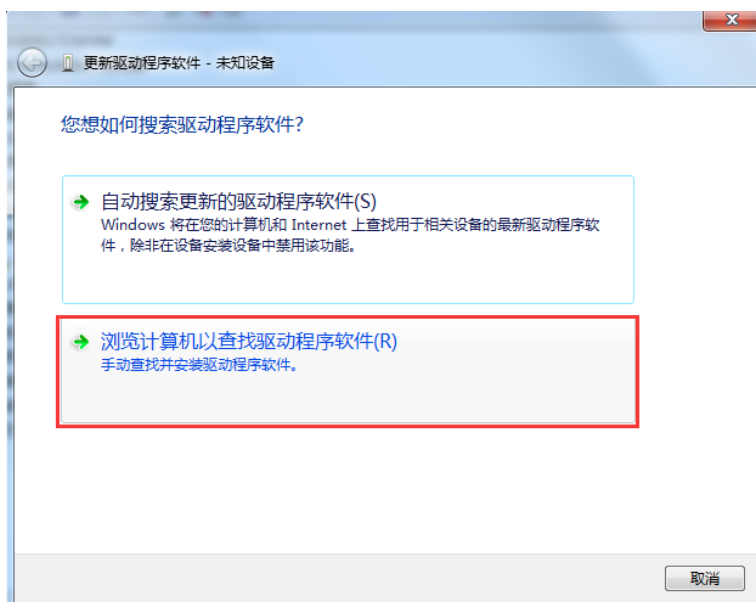


图 1-3-7 驱动更新选择界面

3) 最后选择名为“CH341SER_for_64bit_win7”的驱动程序文件, 位于 “Arduino_Nano 板驱动程序”文件夹, 请根据自己的电脑系统型号选择相应的驱动版本!

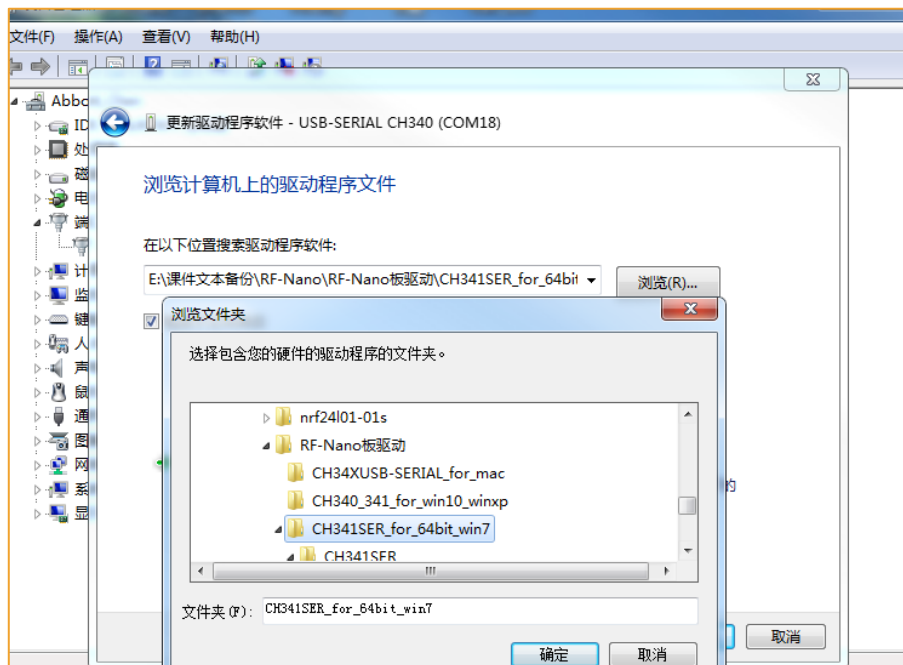


图 1-3-8



图 1-3-9 驱动程序文件选择界面

4) 成功安装之后便会出现下图所示的界面，通知你驱动成功。

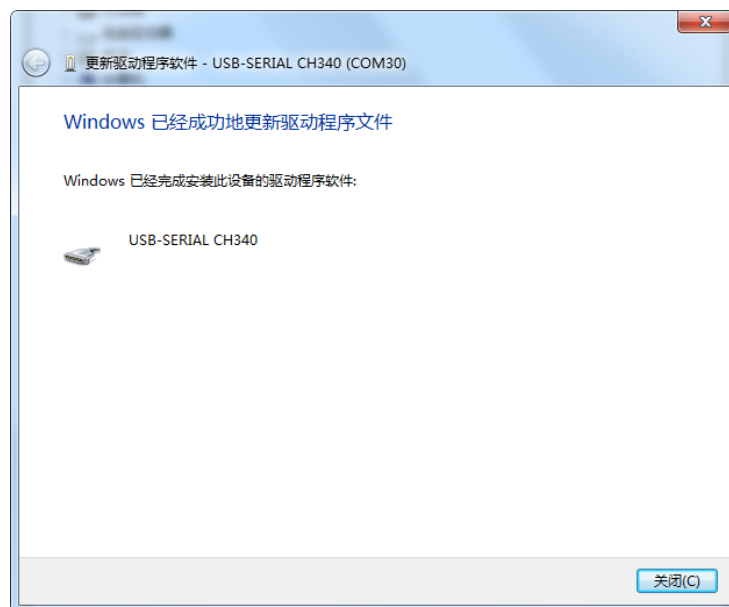


图 1-3-10 驱动安装成功界面

此时，我们再返回“设备管理器”界面，可以看到电脑已成功识别到 Arduino, 如下图 1-3-11 所示。接下来打开 Arduino 编译环境，就可开启 Arduino 之旅了。

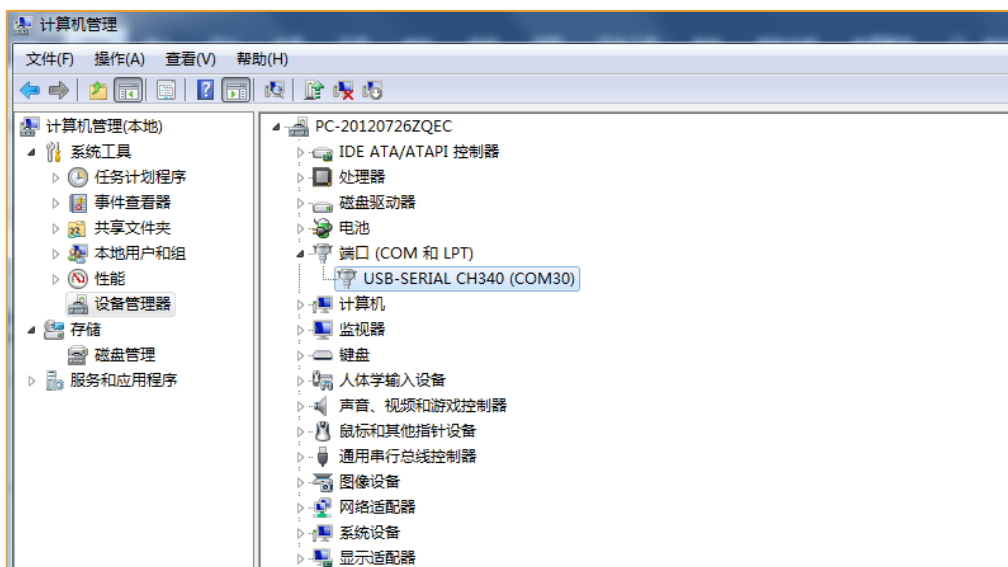


图 1-3-11 驱动成功识别界面

注意：如果安装 CH341SER_for_64bit_win7 失败时，请安装 ft232r_usb_uart。

注意：在 Win10 系统中，部分 Arduino 在接入电脑后（非正版芯片很难被识别），系统会自动下载相对应的驱动，无需自己安装驱动，但是在 Win7 系统中，就需要按照上述的步骤手动安装驱动。

如上图所示我们可以看到 USB 串口被识别为 COM15，但不同的电脑可能不一样，你的可能是 COM4、COM5 等，但是 Arduino Nano 这个一定是一样的。如果没找到 USB 串口，则有可能是你安装有误，或者系统不兼容。

Windows8 系统驱动安装步骤

如果你的电脑是 Windows8 系统：在安装驱动程序之前，您应该保存您正在编辑的这些文件因为在操作过程中会有几次关机。

- 1) 按下“Windows 键”+“R”
- 2) 输入 shutdown.exe /r /o /f /t 00
- 3) 点击“确定”按钮。
- 4) 系统将重新启动到“选择一个选项”屏幕
- 5) 从“选择一个选项”屏幕中选择“疑难解答”
- 6) 从“疑难解答”屏幕中选择“高级选项”
- 7) 从“高级选项”中选择“Windows 启动设置”屏幕
- 8) 点击“重新启动”按钮
- 9) 系统将重新启动到“高级启动选项”屏幕
- 10) 选择“禁用驱动程序签名强制”
- 11) 一旦系统启动，您可以安装 Arduino 驱动程序与 Windows7 相同

如果你的电脑是 WindowsXP 系统:那安装步骤基本和 Windows7 相同，可参考上面的 Windows7 安装步骤。

第二章 RF-NANO 工作原理

2.1 RF-NANO 工作原理介绍

RF-NANO 可以发射数据也可以接收数据。

发射数据时:首先将NRF24L01+配置为发射模式：接着把接收节点地址TX_ADDR和有效数据TX_PLD按照时序由SPI口写入NRF24L01+缓存区，TX_PLD必须在CSN为低时连续写入，而TX_ADDR在发射时写入一次即可，然后CE置为高电平并保持至少10 μ s，延迟130 μ s后发射数据；若自动应答开启，那么NRF24L01+在发射数据后立即进入接收模式，接收应答信号（自动应答接收地址应该与接收节点地址TX_ADDR一致）。如果收到应答，则认为此次通信成功，TX_DS置高，同时TX_PLD从TX_FIFO中清除；若未收到应答，则自动重新发射该数据（自动重发已开启），若重发次数（ARC）达到上限，MAX_RT置高，TX_FIFO中数据保留以便在次重发；MAX_RT或TX_DS置高时，使IRQ变低，产生中断，通知MCU。最后发射成功时，若CE为低则NRF24L01+进入空闲模式1；若发送堆栈中有数据且CE为高，则进入下一次发射；若发送堆栈中无数据且CE为高，则进入空闲模式2。

接收数据时:首先将NRF24L01+配置为接收模式，接着延迟130 μ s进入接收状态等待数据的到来。当接收方检测到有效的地址和CRC时，就将数据包存储在RX_FIFO中，同时中断标志位RX_DR置高，IRQ变低，产生中断，通知MCU去取数据。若此时自动应答开启，接收方则同时进入发射状态回传应答信号。最后接收成功时，若CE变低，则NRF24L01+进入空闲模式1。

在写寄存器之前一定要进入待机模式或掉电模式。如图 2-1-1 所示，给出 SPI 操作及时序图：

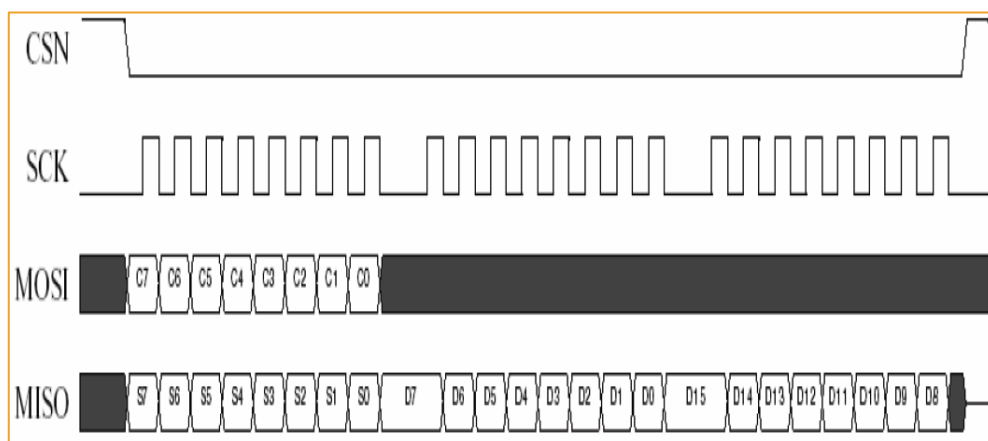


图 2-1-1 SPI 读操作

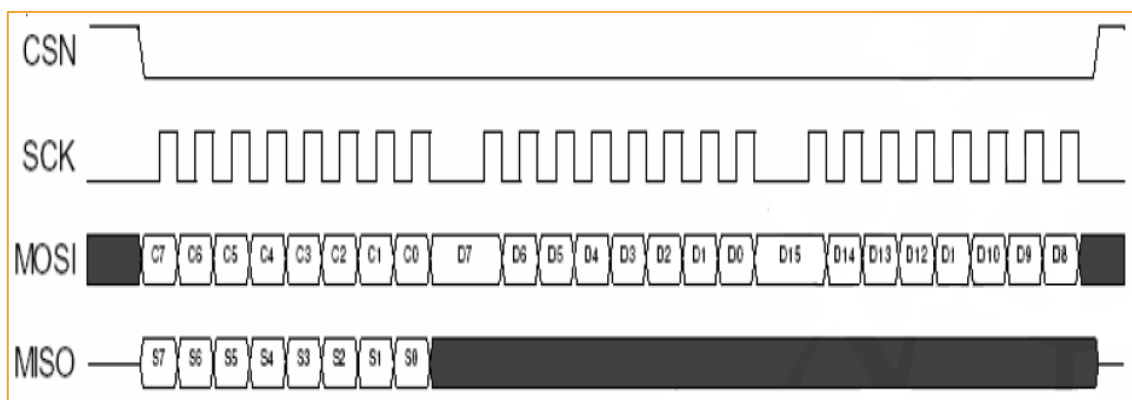


图 2-1-1 SPI 写操作

2.2 配置字

SPI 口为同步串行通信接口，最大传输速率为 10 Mb/s，传输时先传送低位字节，再传送高位字节。但针对单个字节而言，要先送高位再送低位。与 SPI 相关的指令共有 8 个，使用时这些控制指令由 NRF24L01+的 MOSI 输入。相应的状态和数据信息是从 MISO 输出给 MCU。

nRF24L01+所有的配置字都由配置寄存器定义，这些配置寄存器可通过 SPI 口访问。NRF24L01+ 的配置寄存器共有 25 个，常用的配置寄存器如表 2 所示。

表2：常用配置寄存器

地址 (H)	寄存器名称	功能
00	CONFIG	设置NRF24L01+工作模式
01	EN_AA	设置接收通道及自动应答
02	EN_RXADDR	使能接收通道地址
03	SETUP_AW	设置地址宽度
04	SETUP_RETR	设置自动重发数据时间和次数
07	STATUS	状态寄存器，用来判定工作状态
0A~0F	RX_ADDR_P0~P5	设置接收通道地址
10	TX_ADDR	设置接收接点地址
11~16	RX_PW_P0~P5	设置接收通道的有效数据宽度

2.3 RF-NANO 的工作模式

NRF24L01+ 芯片内部有状态机，控制着芯片在不同工作模式之间的转换。NRF24L01+ 可配置为 Shutdown、Standby、Idle-TX、TX 和 RX 五种工作模式。状态转换图如图 2-3-1 所示。

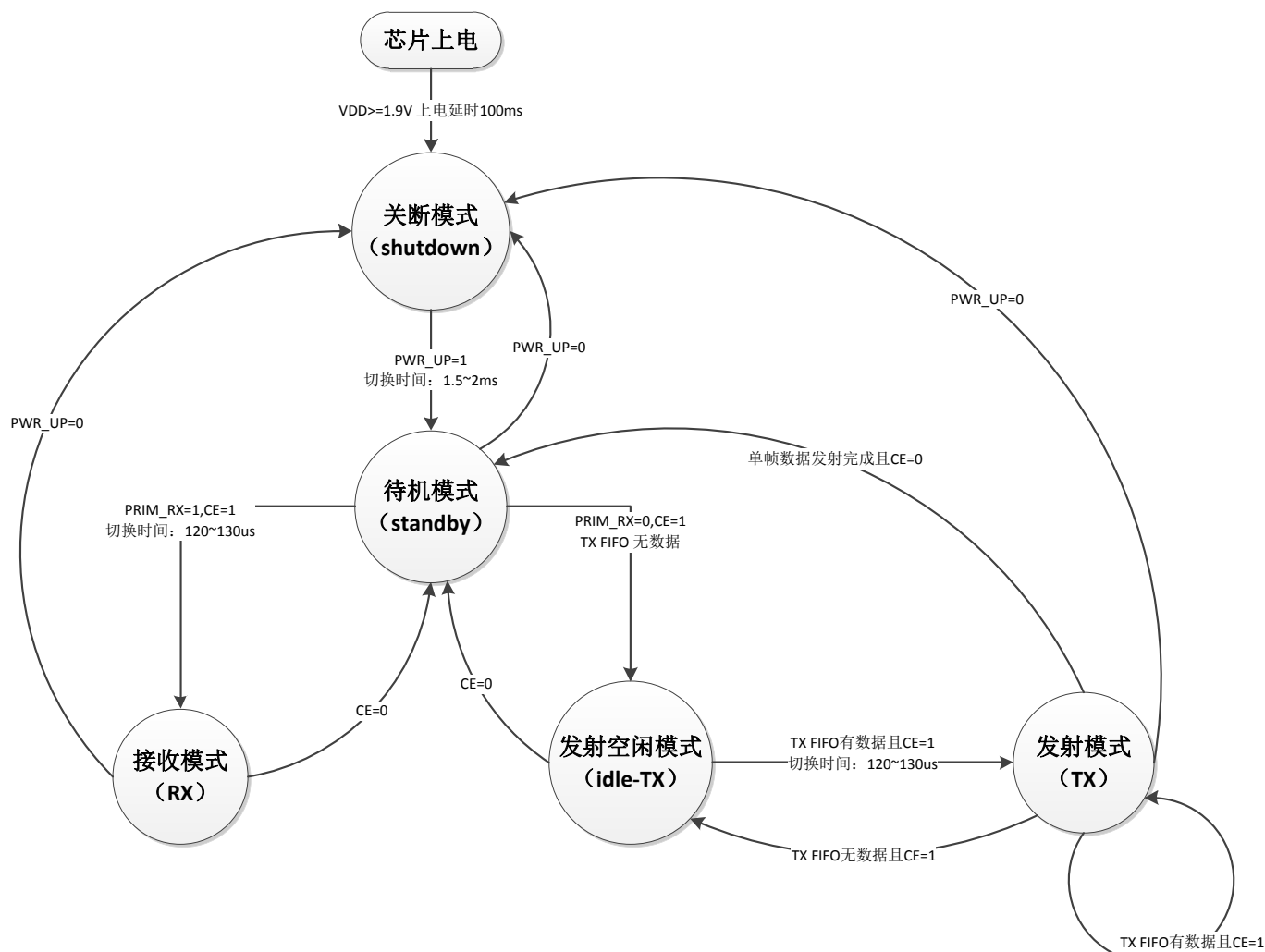


图 2-3-1

Shutdown 工作模式：

在 Shutdown 工作模式下，NRF24L01+ 所有收发功能模块关闭，芯片停止工作，消耗电流最小，但所有内部寄存器值和 FIFO 值保持不变，仍可通过 SPI 实现对寄存器的读写。设置 CONFIG 寄存器的 PWR_UP 位的值为 0，芯片立即返回到 Shutdown 工作模式。

Standby 工作模式：

在 Standby 工作模式，只有晶体振荡器电路工作，保证了芯片在消耗较少电流的同时能够快速启动。设置 CONFIG 寄存器下的 PWR_UP 位的值为 1，芯片待时钟稳定后进入 Standby 模式。芯片的时钟稳定时间一般为 1.5~2ms，与晶振的性能有关。当引脚 CE=1 时，芯片将由 Standby 模式

进入到 Idle-TX 或 RX 模式, 当 CE=0 时, 芯片将由 Idle-TX、TX 或 RX 模式返回到 Standby 模式。

Idle-TX 工作模式:

在 Idle-TX 工作模式下, 晶体振荡器电路及时钟电路工作。相比于 Standby 模式, 芯片消耗更多的电流。当发送端 TX FIFO 寄存器为空, 并且引脚 CE=1 时, 芯片进入到 Idle-TX 模式。在该模式下, 如果有新的数据包被送到 TX FIFO 中, 芯片内部的电路将立即启动, 切换到 TX 模式将数据包发送。在 Standby 和 Idle-TX 工作模式下, 所有内部寄存器值和 FIFO 值保持不变, 仍可通过 SPI 实现对寄存器的读写。

TX 工作模式:

当需要发送数据时, 需要切换到 TX 工作模式。芯片进入到 TX 工作模式的条件为: TX FIFO 中有数据, CONFIG 寄存器的 PWR_UP 位的值为 1, PRIM_RX 位的值为 0, 同时要求引脚 CE 上有一个至少持续 10us 的高脉冲。芯片不会直接由 Standby 模式直接切换到 TX 模式, 而是先立即切换到 Idle-TX 模式, 再由 Idle-TX 模式自动切换到 TX 模式。Idle-TX 模式切换到 TX 模式的时间为 120us~130us 之间, 但不会超过 130us。单包数据发送完成后, 如果 CE=1, 则由 TX FIFO 的状态来决定芯片所处的工作模式, 当 TX FIFO 还有数据, 芯片继续保持在 TX 工作模式, 并发送下一包数据; 当 TX FIFO 没有数据, 芯片返回 Idle-TX 模式; 如果 CE=0, 立即返回 Standby 模式。数据发射完成后, 芯片产生数据发射完成中断。

RX 工作模式:

当需要接收数据时, 需要切换到 RX 工作模式。芯片进入到 RX 工作模式的条件为: 设置寄存器 CONFIG 的 PWR_UP 位的值为 1, PRIM_RX 位的值为 1, 并且引脚 CE=1。芯片由 Standby 模式切换到 RX 模式的时间为 120~130us。当接收到数据包的地址与芯片的地址相同, 并且 CRC 检查正确时, 数据会自动存入 RX FIFO, 并产生数据接收中断。芯片最多可以同时存三个有效数据包, 当 FIFO 已满, 接收到的数据包被自动丢掉。在接收模式下, 可以通过 RSSI 寄存器检测接收信号功率。当接收到的信号强度大于 -60dBm 时, RSSI 寄存器的 RSSI 位的值将被设置为 1。否则, RSSI=0。RSSI 寄存器的更新方法有两种: 当接收到有效的数据包后, RSSI 会自动更新, 此外, 将芯片从 RX 模式换到 Standby 模式时 RSSI 也会自动更新。RSSI 的值会随温度的变化而变化, 范围在 $\pm 5\text{dBm}$ 以内。

3.1 实现两个 RF-NANO 点对点通讯

3.1.1 连线方式

准备两个 RF-NANO 或者一个 RF-NANO 和 NRF24L01+模块，以及一个 Arduino UNO R3(Arduino Nano V3.0)，如图 3-1-1 所示。

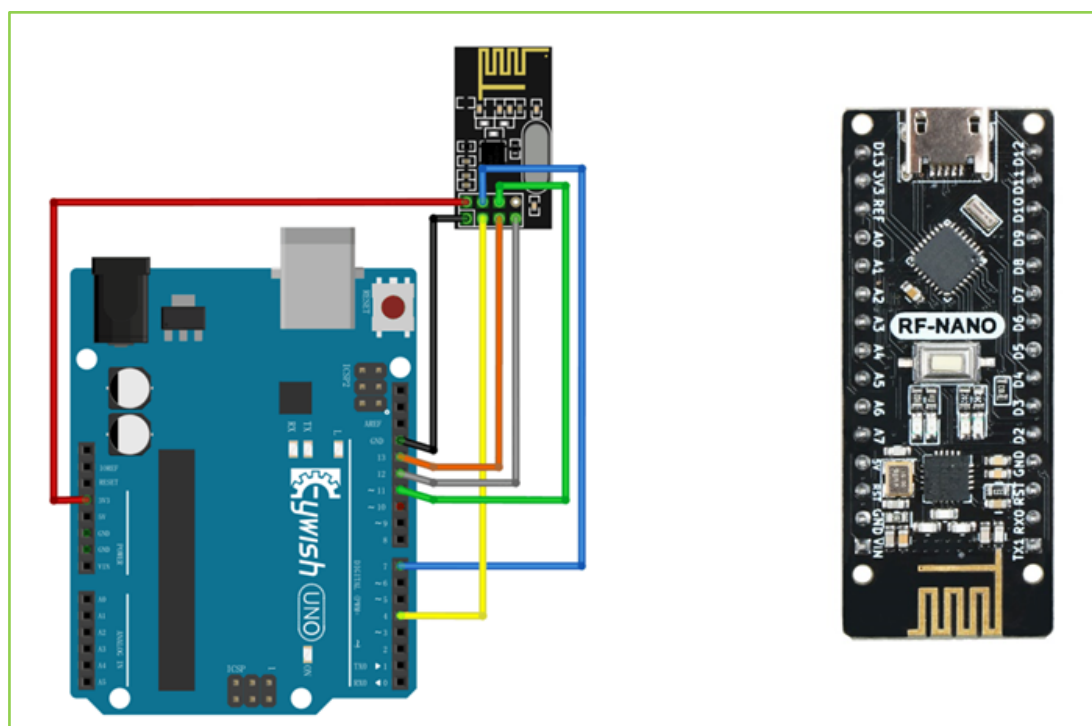
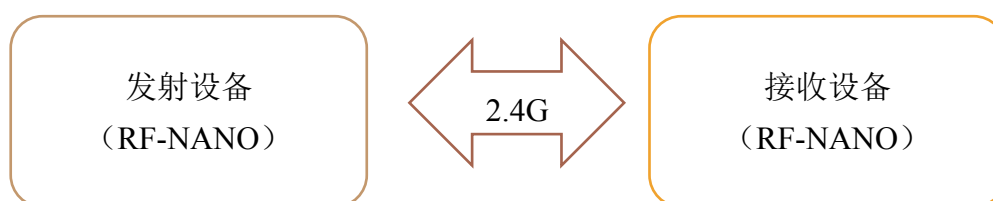


图 3-1-1

Arduino 和 NRF24L01+接线方式

arduino Uno	NRF24L01+
-------------	-----------

+3.3V	VCC
GND	GND
7	4pin CSN
4	3pin CE
11pin	6pin MOSI
12pin	7pin MISO
13pin	5pin SCK

3.1.2 程序原理

发射流程:

- 1、首先将 NRF24L01+配置为发射模式，
- 2、接着把接收端的地址 TX_ADDR 和要发送的数据 TX_PLD 按照时序由 SPI 口写入 NRF24L01+缓存区。
- 3、Arduino 将 CE 配置为高电平并保持至少 10 μ s，延迟 130 μ s 后发射数据；若自动应答开启，那么 NRF24L01+在发射数据后立即进入接收模式，接收应答信号。如果收到应答，则认为此次通信成功。
- 4、NRF24L01+会自动将 TX_DS 置高，同时 TX_PLD 从发送堆栈中清除；若未收到应答，则自动重新发射该数据，若重发次数 (ARC_CNT) 达到上限，MAX_RT 置高，TX_PLD 不会被清除；MAX_RT 或 TX_DS 置高时，IRQ 变低触发 MCU 中断。最后发射成功时，若 CE 为低，则 NRF24L01+进入待机模式
- 5、若发送堆栈中有数据且 CE 为高，则进入下一次发射；若发送堆栈中无数据且 CE 为高，则进入发射空闲模式。

接收数据流程:

- 1、NRF24L01+接收数据时，首先将 NRF24L01+配置为接收模式，
- 2、接着延迟 130 μ s 进入接收状态等待数据的到来。当接收方检测到有效的地址和 CRC 时，就将数据包存储在接收堆栈中，同时中断标志位 RX_DR 置高，IRQ 变低，以便通知 MCU 去取数据。
- 3、若此时自动应答开启，接收方则同时进入发射状态回传应答信号。最后接收成功时，若 CE 变低，则 NRF24L01+进入空闲模式 1。

3.1.3 程序代码

发射数据程序代码:

- 代码路径: RF-NANO 实验例程程序\点对点通讯实验例程程序\Emitter\Emitter.ino
- 代码源码

```
#include <SPI.h>
#include "Mirf.h"
```

```
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"
Nrf24l Mirf = Nrf24l(10, 9);
byte value;

void setup()
{
  Serial.begin(9600);
  Mirf.spi = &MirfHardwareSpi;
  Mirf.init();
  //Set your own address (sender address) using 5 characters
  Mirf.setRADDR((byte *)"ABCDE");
  Mirf.payload = sizeof(value);
  Mirf.channel = 90;           //Set the channel used
  Mirf.config();
}

void loop()
{
  Mirf.setTADDR((byte *)"FGHIJ");           //Set the receiver address
  value = random(255);                       //0-255 random number
  Mirf.send(&value);                         //Send instructions, send random number value
  Serial.print("Wait for sending.....");
  while (Mirf.isSending())//Until you send successfully, exit the loop
  delay(1);
  Serial.print("Send success:");
  Serial.println(value);
  delay(1000);
}
```

发射设备发送的数据:

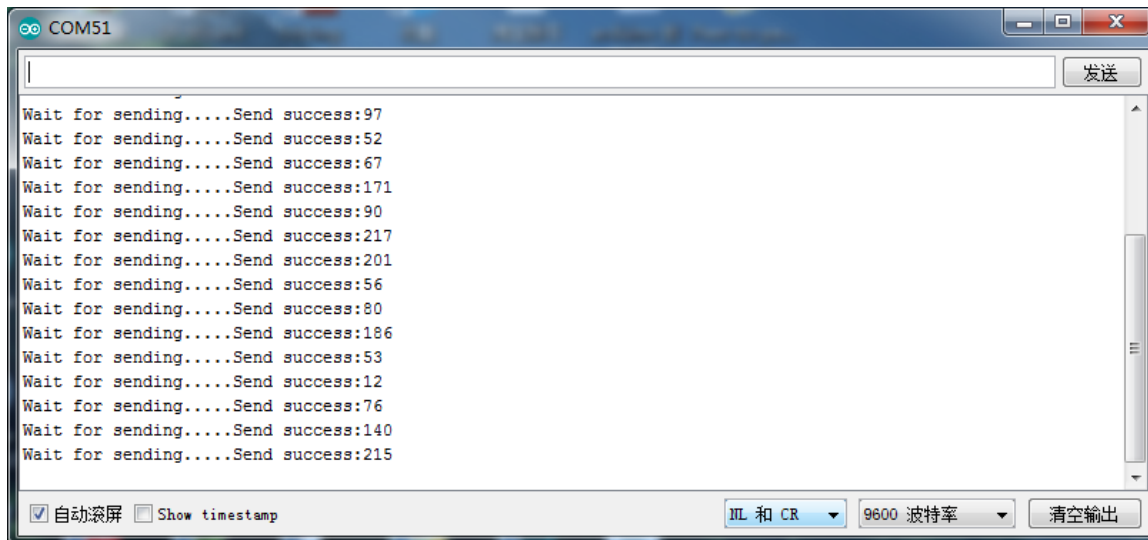


图 3-1-2

接收设备程序代码:

- 代码路径: RF-NANO 实验例程程序\点对点通讯实验例程程序\Receive\ Receive.ino
- 代码源码

```
//Receiver program
#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"
Nrf24l Mirf = Nrf24l(10, 9);
byte value;
void setup()
{
    Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.init();
    Mirf.setRADDR((byte *)"FGHIJ"); //Set your own address (receiver address) using
5 characters
    Mirf.payload = sizeof(value);
    Mirf.channel = 90; //Set the used channel
    Mirf.config();
    Serial.println("Listening..."); //Start listening to received data
}
```



```
void loop()
{
  if (Mirf.dataReady()) { //When the program is received, the received data is output
from the serial port
    Mirf.getData(&value);
    Serial.print("Got data: ");
    Serial.println(value);
  }
}
```

接收设备接收的数据:

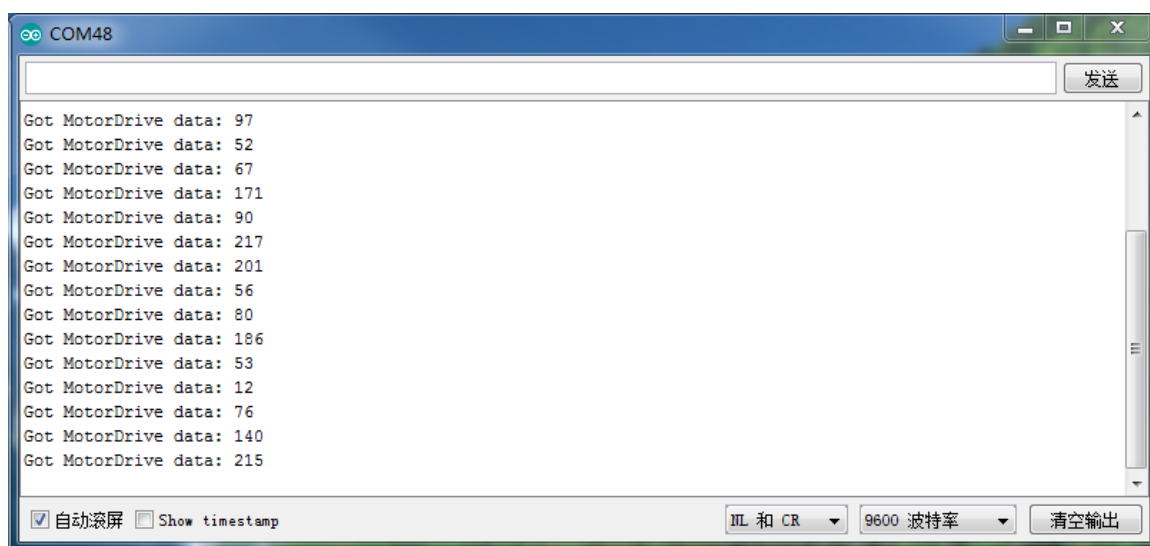
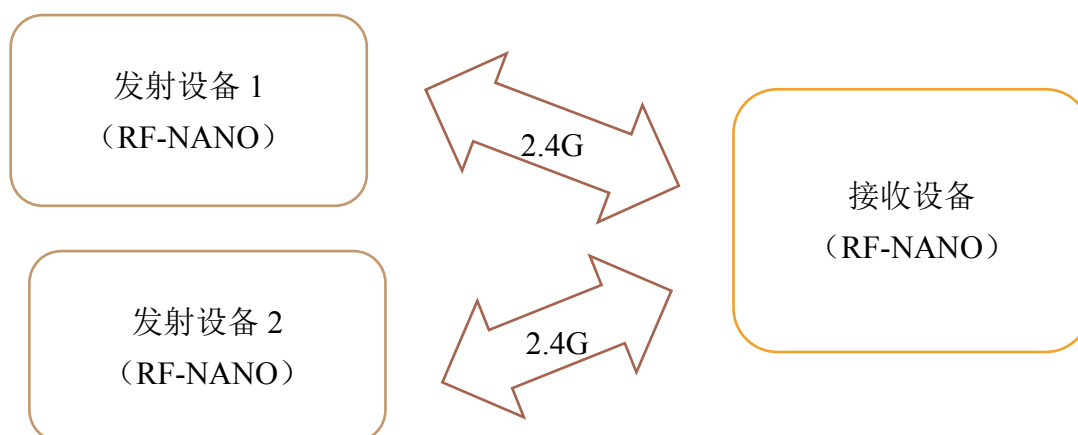


图 3-1-2

3.2 实现多个发送对一个接收通讯

3.2.1 实验原理框图



3.2.2 程序代码

发射设备 1 程序代码：

- 代码路径：RF-NANO 实验例程程序\多个发送对一个接收通讯实验例程程序
\\Emmitter1\\Emmitter1.ino
- 代码源码

```
#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"
Nrf24l Mirf = Nrf24l(10, 9);
int value;

void setup()
{
    Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.init();
    //Set your own address (sender address) using 5 characters
    Mirf.setRADDR((byte *)"ABCDE");
    Mirf.payload = sizeof(value);
    Mirf.channel = 10;           //Set the channel used
    Mirf.config();
}

void loop()
{
    Mirf.setTADDR((byte *)"FGHIJ");           //Set the receiver address
    value = 100;
    Mirf.send((byte *)&value);               //Send instructions, send random number value
    Serial.print("Wait for sending.....");
    while (Mirf.isSending()) delay(1);         //Until you send successfully, exit the loop
    Serial.print("Send success:");
    Serial.println(value);
    delay(1000);
}
```

发射设备 1 发送的数据:

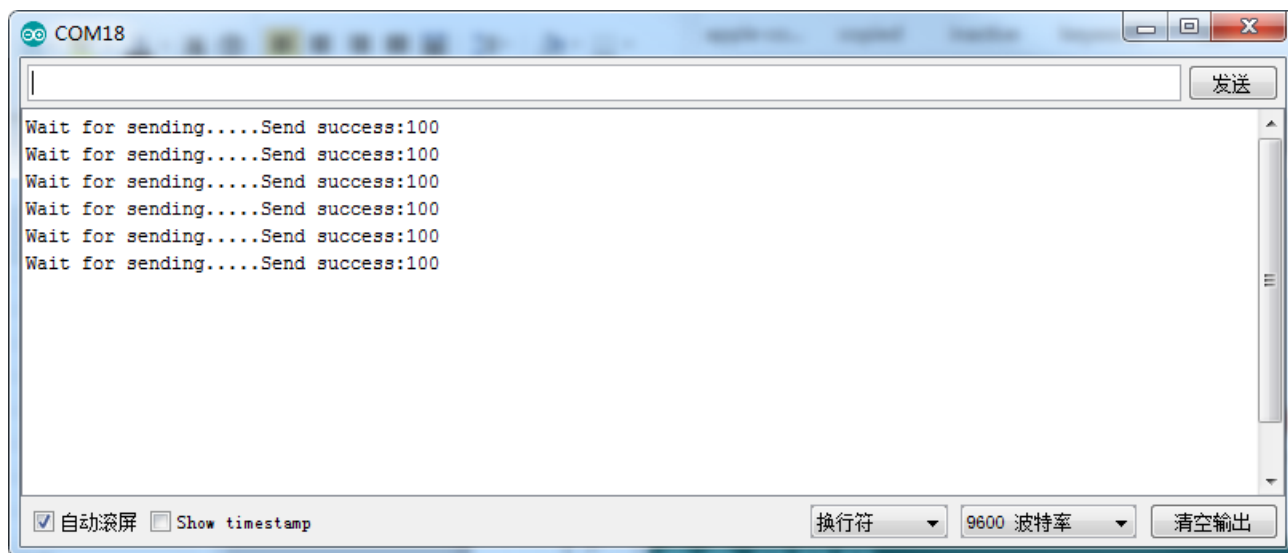


图 3-2-1

发射设备 2 程序代码:

- 代码路径: RF-NANO 实验例程程序\多个发送对一个接收通讯实验例程程序\Emmitter2\Emmitter2.ino
- 代码源码

```
//Transmitter program

#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"
Nrf24l Mirf = Nrf24l(10, 9);
int value;

void setup()
{
  Serial.begin(9600);
  Mirf.spi = &MirfHardwareSpi;
  Mirf.init();
  //Set your own address (sender address) using 5 characters
  Mirf.setRADDR((byte *) "ABCDE");
  Mirf.payload = sizeof(value);
  Mirf.channel = 20;           //Set the channel used
  Mirf.config();
}
```

```

}

void loop()
{
    Mirf.setTADDR((byte *)"FGHIJ");           //Set the receiver address
    value = 200;                               //0-255 random number
    Mirf.send((byte *)&value);                 //Send instructions, send random number value
    Serial.print("Wait for sending.....");
    while (Mirf.isSending()) delay(1);         //Until you send successfully, exit the loop
    Serial.print("Send success:");
    Serial.println(value);
    delay(1000);
}

```

发射设备 2 发送的数据:

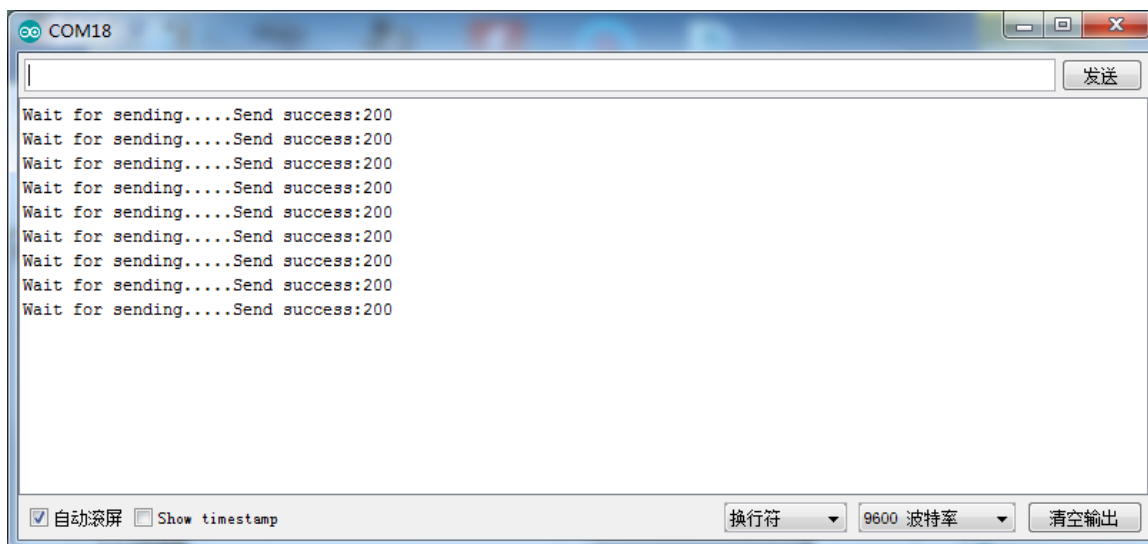


图 3-2-2

接收数据程序代码:

- 代码路径: RF-NANO 实验例程程序\多个发送对一个接收通讯实验例程程序\All_Receive\All_Receive.ino
- 代码源码

```

//Receiver program
#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"

```

```
#include "MirfHardwareSpiDriver.h"

Nrf24l Mirf = Nrf24l(10, 9);

int value;

void setup()
{
    Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.init();
    Mirf.setRADDR((byte *)"FGHIJ"); //Set your own address (receiver address) using
5 characters
    Mirf.payload = sizeof(value);
    Mirf.channel = 10;           //Set the used channel
    Mirf.config();
    Serial.println("Listening..."); //Start listening to received data
}

void loop()
{
    Mirf.ceLow();
    Mirf.configRegister(RF_CH, 10); //switch channel 10
    Mirf.ceHi();
    if (Mirf.dataReady()) { //When the program is received, the received data is output
from the serial port
        Mirf.getData((byte *) &value);
        Serial.print("Got Emitter1 data is: ");
        Serial.println(value);
    }
    delay(1000);

    Mirf.ceLow();
    Mirf.configRegister(RF_CH, 20); //switch channel 20
    Mirf.ceHi();
    if (Mirf.dataReady()) { //When the program is received, the received data is output
from the serial port
        Mirf.getData((byte *) &value);
```

```
Serial.print("Got Emitter2 data is: ");
Serial.println(value);
}
delay(1000);
}
```

接收设备接收的数据:

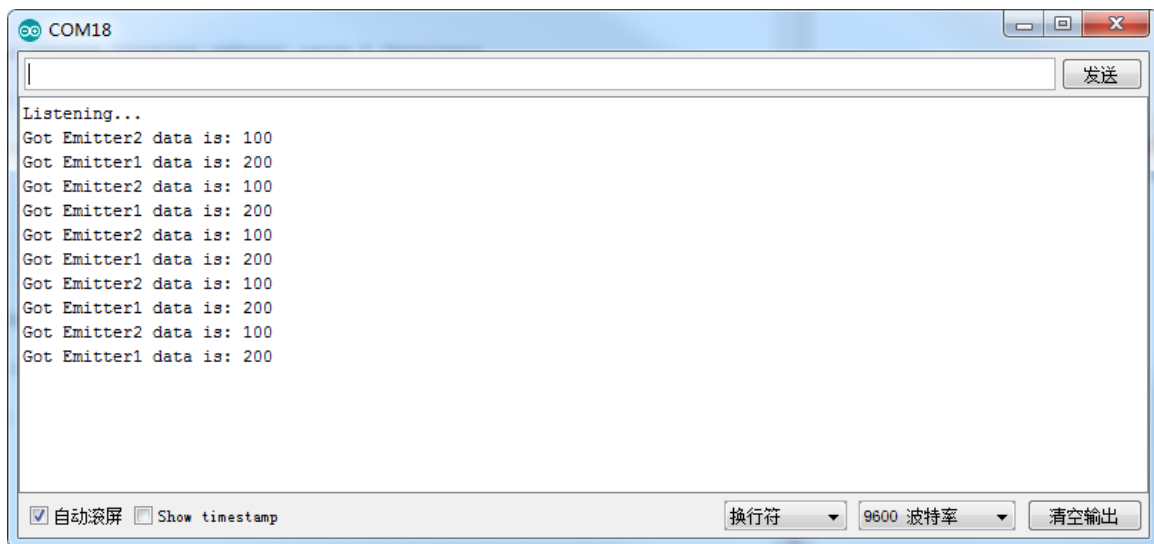
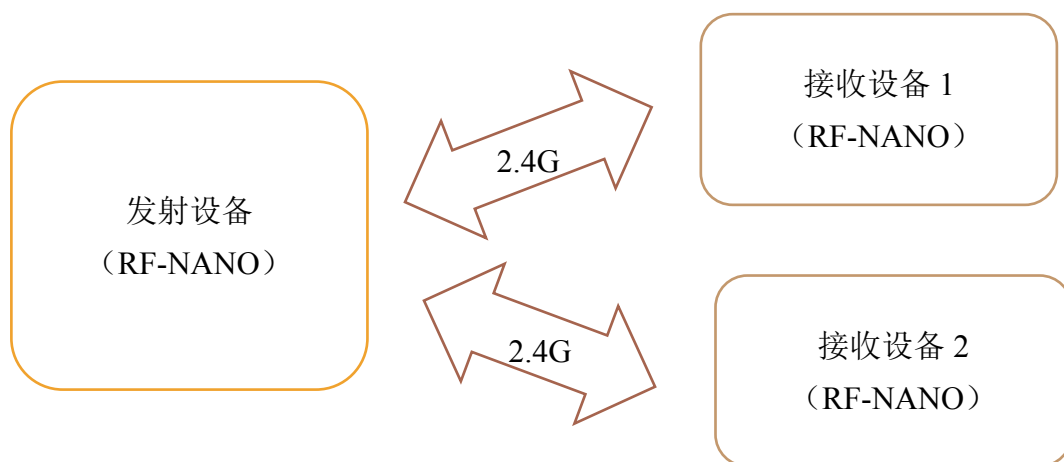


图 3-2-3

3.3 实现一个发送对多个接收通讯

3.3.1 实验原理框图



3.3.2 程序代码

发射数据程序代码：

- 代码路径：RF-NANO 实验例程程序\一个发送对多个接收通讯实验例程程序\ All_Emitter \All_Emitter.ino
- 代码源码

```
//Transmitter program

#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"
Nrf24l Mirf = Nrf24l(10, 9);
int value1,value2;

void setup()
{
    Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.init();
    //Set your own address (sender address) using 5 characters
    Mirf.setRADDR((byte *)"ABCDE");
}

void loop()
{
    Mirf.payload = sizeof(value1);
    Mirf.channel = 10;           //Set the channel used
    Mirf.config();
    if(Mirf.channel == 10)
    {
        Mirf.setTADDR((byte *)"FGHIJ");           //Set the receiver address
        value1 = 100;
        Mirf.send((byte *)&value1);           //Send instructions, send random number value
        Serial.print("Wait for sending.....");
        while (Mirf.isSending()) delay(1);           //Until you send successfully, exit the loop
        Serial.print("value1 Send success:");
        Serial.println(value1);
        delay(500);
    }
}
```



```

}
Mirf.payload = sizeof(value2);
Mirf.channel = 20;           //Set the channel used
Mirf.config();
if(Mirf.channel == 20)
{
Mirf.setTADDR((byte *) "KLMNO");           //Set the receiver address
value2 = 200;
Mirf.send((byte *)&value2);           //Send instructions, send random number value
Serial.print("Wait for sending.....");
while (Mirf.isSending()) delay(1);           //Until you send successfully, exit the loop
Serial.print("value2 Send success:");
Serial.println(value2);
delay(500);
}
}

```

发射设备发送的数据:

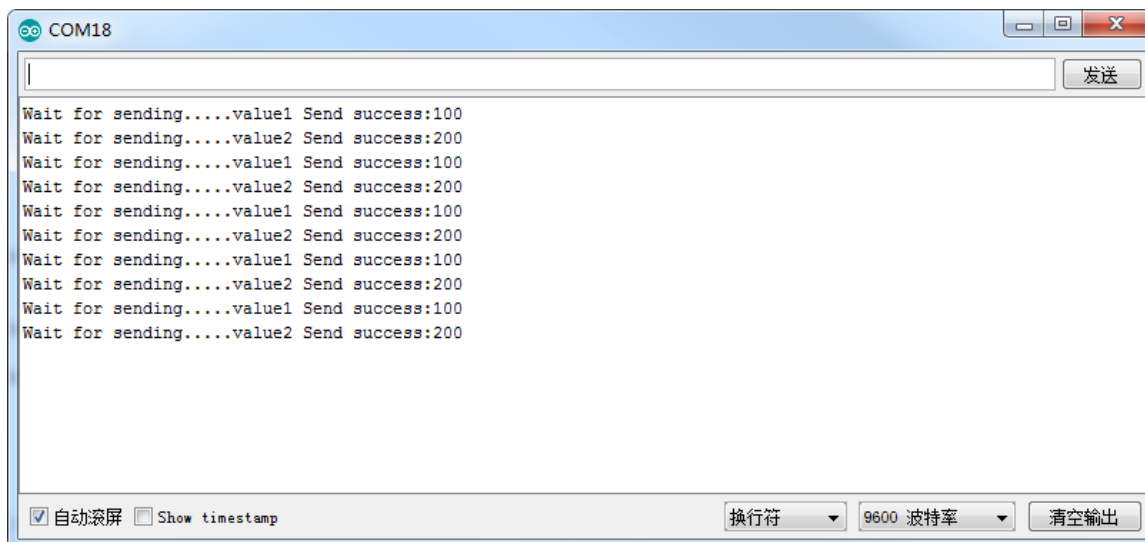


图 3-3-1

接收设备 1 代码:

- 代码路径: RF-NANO 实验例程程序\一个发送对多个接收通讯实验例程程序\Receive1\Receive1.ino
- 代码源码

```

//Receiver program
#include <SPI.h>

```

```
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"

Nrf24l Mirf = Nrf24l(10, 9);

int value;

void setup()
{
    Serial.begin(9600);
    Mirf.spi = &MirfHardwareSpi;
    Mirf.init();
    Mirf.setRADDR((byte *)"FGHIJ"); //Set your own address (receiver address) using 5
characters
    Mirf.payload = sizeof(value);
    Mirf.channel = 10;           //Set the used channel
    Mirf.config();
    Serial.println("Listening..."); //Start listening to received data
}

void loop()
{
    Mirf.ceLow();
    Mirf.configRegister(RF_CH, 10); //switch channel 10
    Mirf.ceHi();
    if (Mirf.dataReady()) { //When the program is received, the received data is output from
the serial port
        Mirf.getData((byte *) &value);
        Serial.print("Receive1 got data is: ");
        Serial.println(value);
    }
    delay(800);
}
```

接收设备 1 接收的数据:

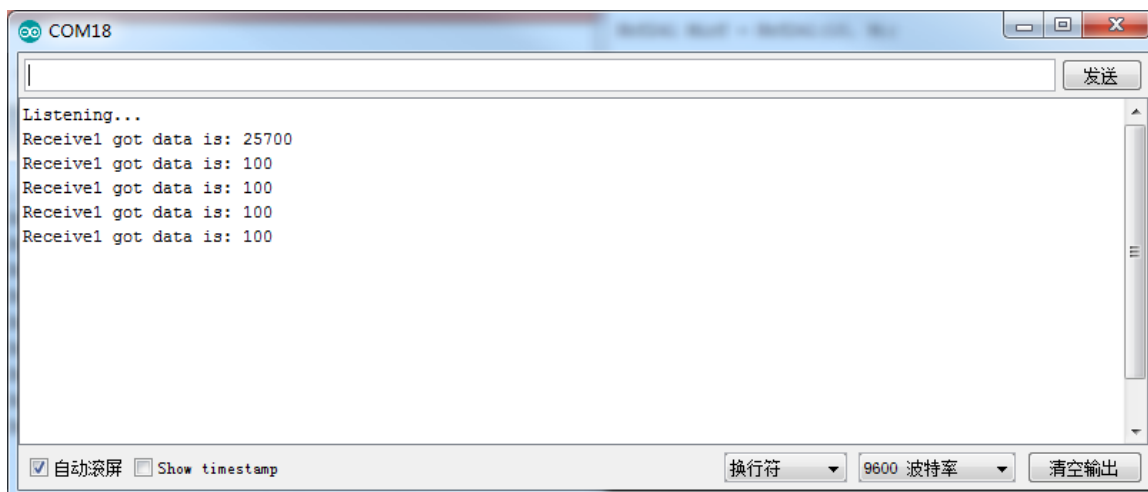


图 3-3-2

接收设备 2 程序代码:

- 代码路径: RF-NANO 实验例程程序\一个发送对多个接收通讯实验例程程序\Receive2\Receive2.ino
- 代码源码

```
//Receiver program
#include <SPI.h>
#include "Mirf.h"
#include "nRF24L01.h"
#include "MirfHardwareSpiDriver.h"

Nrf24l Mirf = Nrf24l(10, 9);

int value;

void setup()
{
  Serial.begin(9600);
  Mirf.spi = &MirfHardwareSpi;
  Mirf.init();
  Mirf.setRADDR((byte *)"KLMNO"); //Set your own address (receiver address) using 5
  characters
  Mirf.payload = sizeof(value);
  Mirf.channel = 20; //Set the used channel
  Mirf.config();
}
```

```
Serial.println("Listening..."); //Start listening to received data
}

void loop()
{
    Mirf.ceLow();
    Mirf.configRegister(RF_CH, 20); //switch channel 20
    Mirf.ceHi();
    if (Mirf.dataReady()) { //When the program is received, the received data is output
from the serial port
        Mirf.getData((byte *) &value);
        Serial.print("Receive2 got data is: ");
        Serial.println(value);
    }
    delay(1000);
}
```

接收设备 2 接收的数据:

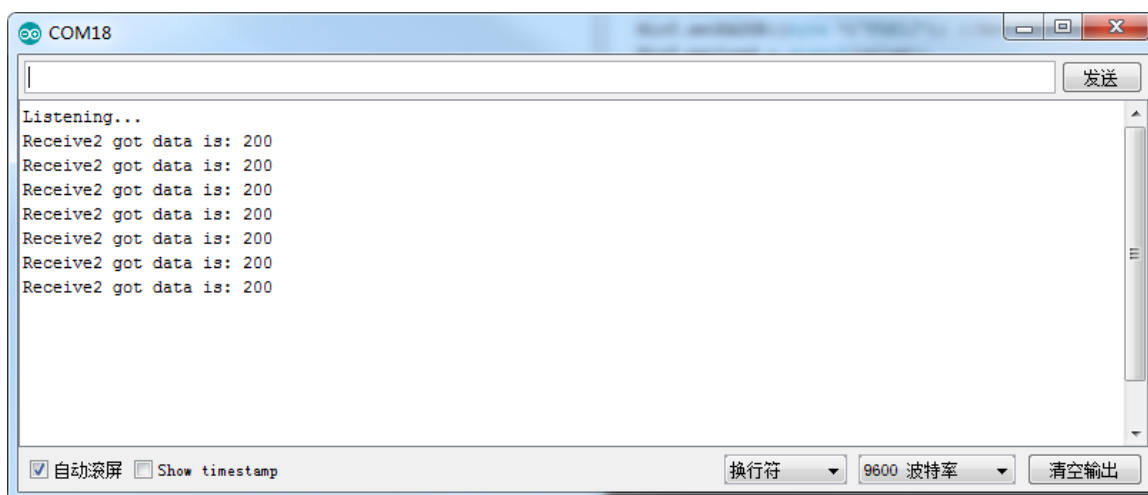


图 3-3-3

第四章 设置 RF-NANO 发射功率和数据发送速率

4.1 设置 RF-NANO 发射功率

RF-NANO发射功率共有 4 个等级，分别是：-18dBm, -12dBm, -6dBm, 0dBm;发射功率越大，传输距离越远；发射功率可以在软件中设置，打开 RF-NANO 实验例程程序,在 Mirf.cpp 中有设置发射功率的函数：SetOutputRF_PWR(uint8_t val)，可设置不同的发射功率，注意需要在关断模式下设置寄存器值，所以在 config()这个函数中调用。

```
void Nrf24l::SetOutputRF_PWR(uint8_t val) //Set tx power :
0=-18dBm,1=-12dBm,2=-6dBm,3=0dBm,
{
    configRegister(RF_SETUP, (val<< RF_PWR) );
}
void Nrf24l::config()
// Sets the important registers in the MiRF module and powers the module
// in receiving mode
// NB: channel and payload must be set now.
{
    configRegister(RF_CH, channel);      // Set RF channel
    configRegister(RX_PW_P0, payload);    // Set length of incoming payload
    configRegister(RX_PW_P1, payload);
    SetSpeedDataRates(0);                //Select between the high speed data rates:250Kbps
    powerUpRx();                          // Start receiver
    flushRx();
}
```

4.2 设置 RF-NANO 数据传输速率

RF-NANO 数据传输速率共有 3 个等级，分别是：1Mbps, 2Mbps, 250Kbps；数据传输速率可以在软件中设置，打开 RF-NANO 实验例程程序,在 Mirf.cpp 中有设置数据传输速率的函数：

SetSpeedDataRates(uint8_t val)，可设置不同的发射功率。注意需要在关断模式下设置寄存器值，所以在 config()这个函数中调用。

```
void Nrf24l::SetSpeedDataRates(uint8_t val) //Select between the high speed data
rates:0=1Mbps, 1=2Mbps, 2=250Kbps
{
    if(val>1)
    {
        configRegister(RF_SETUP, (1 << RF_DR_LOW) );
    }
}
```

```
else
{
    configRegister(RF_SETUP, (val << RF_DR_HIGH) );
}
}
```