Simple nRF24L01+ 2.4GHz transceiver demo

Robin2 ago. '16 post #1

Introduction
The nRF24L01+ 2.4GHz transceiver modules are cheap and very effective but I have come across a few Threads in which people were having trouble with them so I thought it might be useful to gather together in one place a basic step-by-step sequence for getting them to work.

EDIT 03 Feb 2021 - For this Tutorial install Version 1.1.7 of the RF24 library (available with the Arduino Library Manager). More information in Reply #30.

Edit 29 Aug 2016. *I have today downloaded and successfully tested all 3 pairs of programs with Arduino IDE 1.6.3 - 2 small corrections were needed for SimpleTxAckPayload.ino*

Edit 06 Sep ~~2017~~ 2016 (*oops*). *Modified examples 1 and 2 to listen on Pipe 1 in accordance with the text*

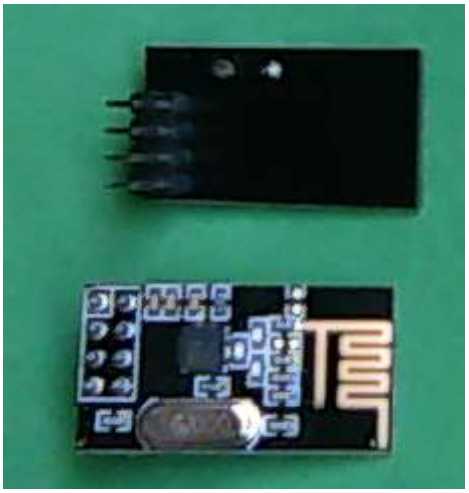
Edit 18 Mar 2017 Added Reply #29 with a simple program to check the connection with the Arduino

TMRh20 version of the RF24 library
This tutorial uses the **TMRh20 version of the RF24 library** which fixes some problems in the earlier ManiacBug version. Unfortunately, however, TMRh20 did not think to give his version a different name so it can be difficult to be certain which one is on your PC. If, before reading this, you have downloaded and installed the RF24 library the simplest thing may be to delete all traces of it and then download and install the TMrh20 version. Note that the demo programs will NOT work with the ManiacBug version of the library.

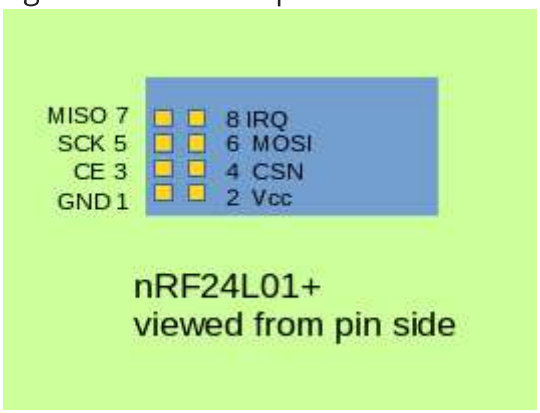
Saltar al contenido principal

nRF24L01+ module pin connections

The nRF24 modules that I am using look like this.



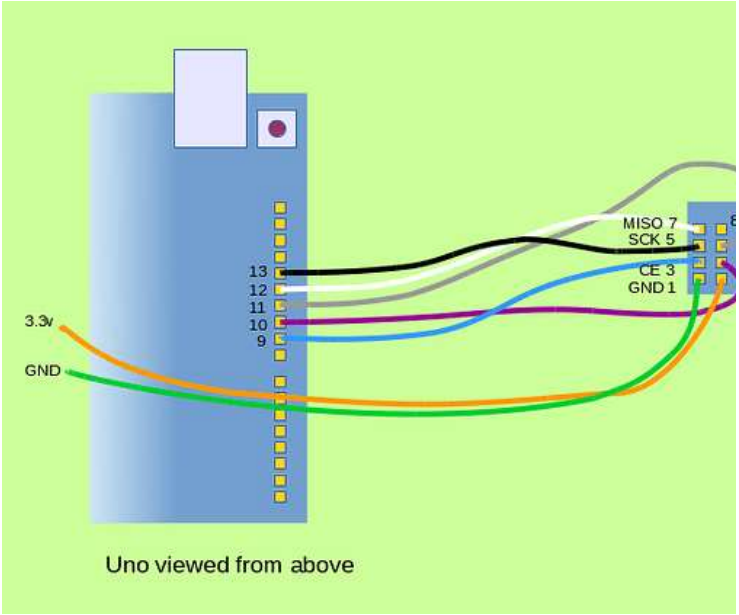
With the pins uppermost and the antenna on the right hand side the pin connections are



Arduino connections

Because the nRF24s use SPI to communicate with the Arduino they must use Arduino pins 13, 12 and 11 (SCK, MISO and MOSI). It is also necessary to connect the CSN and CE pins and any of the Arduino I/O pins can be used for them. However for demo purposes it seems easiest to use pins 10 and 9 so that all 5 connections are adjacent on pins 13 to 9. I use some female-male jumper wires to connect to the nRF24 for test and development. The jumper wires come as a piece of ribbon cable and if you separate a piece with 5 wires and allocate the wires to suit the Arduino pins it is very easy to connect and disconnect the nRF24 from the Arduino without getting the connections mixed up.

Saltar al contenido principal



If you are using pins other than 10 and 9 for CSN and CE you must still set pin 10 for OUTPUT to ensure that the Uno acts as the SPI master.

Powering the nRF24L01+ module
The nRF24 modules require a 3.3v power supply. Be careful NOT to connect the VCC pin to a 5v power supply. I have found that they work satisfactorily when connected to the 3.3v pin on my Unos.

It is recommended to plce a 10µF capacitor across VCC and GND as near to the module as possible and I have found this essential when I made some units with an Atmega328 chip on a breadboard. The capacitor stabilizes the power supply for the module. However I have not found it necessary to use a capacitor with my Uno (or Mega). But if you believe you have followed all the other instructions correctly and are having problems it would certainly be a good idea to install a capacitor.

Resetting the nRF24
It seems that on some occasions the nRF24 does not reset after a new program is uploaded to the Arduino. This can prevent the program from working properly. If you think that is the case then disconnect the Arduino from the USB cable and reconnect it.

Some of the jargon explained
Because the nRF24s are transceivers they can obviously transmit and receive. To try to avoid confusion I will assume that you are using one them (which I will refer to as TX or "master") to transmit and another one (RX or "slave") to receive.

Like a lot of other equipment (WiFi and Bluetooth, for example) the nRF24L01+ modules broadcast on the 2.4GHz band. The precise frequency is determined by the channel that is selected. Both TX and RX must use the same channel. The default channel for the RF24 library is 76.

When the TX sends a message every RX listening on the same channel will receive the message. The TX includes an "address" in the message and the RX will ignore messages that do not have its address. The address is similar in concept to a phone number except that you cannot easily change the number of your phone. The address is a 5 byte number.

The nRF24L01+ modules can listen on any or all of 6 "pipes". Unfortunately in some RF24 demo programs the variable that is used to hold the address is given the name "pipe" even though pipes and addresses are quite different. For the purpose of this tutorial I will only be using one pipe (pipe 1) for listening. Note that pipe 0 is the only writing pipe. It is possible to listen on pipe 0 but it is simpler to leave that excusively for writing.

The 6 pipes allow an nRF24 to listen for messages from 6 other devices with 6 different addresses. But those devices may not transmit at the exact same time.

Radio interference
Note that if 2 or more TXs transmit at the same time on the same channel they will interfere with each other and the message will not be receivable. That is why an NRF24 cannot receive messages on its 6 pipes at the same instant.

Saltar al contenido principal

Messages will also be garbled if (say) a nearby Wifi system transmits at the same time on the exact

same frequency.

However each individual transmission is very short (a few millisecs) so that it is unlikely that the transmission from your TX will overlap with something else.

Edit 14 Oct 2016 -- It has been pointed out that this is not accurate. I am leaving the erroneous text so that people who may already have read it will understand the changes

~~**Data is sent in 32 byte packets**~~

~~The nRF24L01+ modules transmit data in 32 byte packets. If you only want to send a few bytes the RF24 library will automaatically pad out the message with \0 chars. If you attempt to send more than 32 bytes it will be transmitted as a number of packets. For this tutorial I will not be sending more than 32 bytes. Also, I do not intend to cover the dynamic payload feature which allows the user to select a smaller packet size to reduce transmission time.~~

Revised version

Data Packets

The nRF24L01+ modules can transmit a maximum of 32 bytes in a single message. If you need to send more you will need to break it into a number of separate messages. For this tutorial I will not be sending more than 32 bytes.

There are two modes of operation {A} a fixed payload size which defaults to 32 bytes and can be changed with setPayloadSize() and {B} a dynamic payload mode which is chosen with enableDynamicPayloads(). The dynamic payload mode is automatically applied when you choose the ackPayload feature (see Example 2). You can check the payload size with getDynamicPayloadSize().

When using dynamic payloads you must ensure that you read all the bytes that are received or the communication will break down.

Data validation

The nRF24s automatically include sophisticated systems to identify whether the data received matches the data that was sent. If the data is not

received correctly the RX will not show data available() and will not send an acknowledgment. That means the TX will consider the transmission to have failed. You will see in the example programs that the TX is instructed automatically to retry the transmission up to 5 time before giving up. (The max is 15).

In your RX code you can therefore assume that if data is available() it will be correct.

Also note that, unlike Arduino Serial communication, when the RF24 library shows data as available() it means that the entire mesage has been correctly received.

Program style
In the example programs I have tried to use the same style that I used in **Serial Input Basics** and in **Planning and Implementing a Program**
By arranging the parts into small functions it should be easy to incorporate them into other programs.

Continues in next Post

nRF24L01's not communicating

NRF24L01+PA+LNA dont work (Arduino Nano...

Nrf24l01 module not working

Problem with NRF24

Simple signal wire over 5-7m distance

160 más

Robin2 **ago. '16** **post #2**

Simple one way transmission
The following pair of programs sends a simple message "Message N" from the TX to the RX where N is a number that increments from 0 to 9 so that you can see that successive messages are sent and received.

SimpleTx.ino

Saltar al contenido principal

```
// SimpleTx - the master or the transmitte

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>


#define CE_PIN   9
#define CSN_PIN 10

const byte slaveAddress[5] = {'R','x','A',


RF24 radio(CE_PIN, CSN_PIN); // Create a R

char dataToSend[10] = "Message 0";
char txNum = '0';


unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 1000; //


void setup() {

    Serial.begin(9600);

    Serial.println("SimpleTx Starting");

    radio.begin();
    radio.setDataRate( RF24_250KBPS );
    radio.setRetries(3,5); // delay, count
    radio.openWritingPipe(slaveAddress);
}

//====================

void loop() {
    currentMillis = millis();
    if (currentMillis - prevMillis >= txIn
        send();
        prevMillis = millis();
    }
}
```

Saltar al contenido principal

```
//====================

void send() {

    bool rslt;
    rslt = radio.write( &dataToSend, sizeo
        // Always use sizeof() as it gives
        // For example if dataToSend was a

    Serial.print("Data Sent ");
    Serial.print(dataToSend);
    if (rslt) {
        Serial.println("  Acknowledge rece
        updateMessage();
    }
    else {
        Serial.println("  Tx failed");
    }
}

//================

void updateMessage() {
        // so you can see that new data is
    txNum += 1;
    if (txNum > '9') {
        txNum = '0';
    }
    dataToSend[8] = txNum;
}
```

SimpleRx.ino

```
// SimpleRx - the slave or the receiver

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define CE_PIN   9
#define CSN_PIN 10

const byte thisSlaveAddress[5] = {'R','x',

RF24 radio(CE_PIN, CSN_PIN);
```

Saltar al contenido principal

```
char dataReceived[10]; // this must match
bool newData = false;

//===========

void setup() {

    Serial.begin(9600);

    Serial.println("SimpleRx Starting");
    radio.begin();
    radio.setDataRate( RF24_250KBPS );
    radio.openReadingPipe(1, thisSlaveAddr
    radio.startListening();
}

//=============

void loop() {
    getData();
    showData();
}

//==============

void getData() {
    if ( radio.available() ) {
        radio.read( &dataReceived, sizeof(
        newData = true;
    }
}

void showData() {
    if (newData == true) {
        Serial.print("Data received ");
        Serial.println(dataReceived);
        newData = false;
    }
}
```

Edit 20 Jul 2018 ...
When working properly the Rx program should show "**Data received Message n**" in which n varies from 0 to 9. This should be printed at about

one second intervals.
if all you see is **"Data received"** repeating much more quickly then there is a problem - most likely theArduino is not communicating properly with its nRF24. See Reply #29 for more info

Continues in next Post

Problem with NRF24L01 receiver with Arduin...
Connecting between two NRF24s
NRF24L01+ has <50ft of range
Issue with changing values in nRF24L01
8 bit arduino to 32 bit using nrf24 radio
11 más

Robin2 **ago. '16** **post #3**

Two-way transmission
If the slave is required to send data back to the master it would seem necessary for the two devices to swap roles. Among other things that will mean that the "master" will need an address as well as the slave. The question of timing also needs to be considered - for example, how long should the master continue listening for the reply from the slave? And, if the slave is trying to write rather than listen it may miss a transmission from the master.

Two-way transmission using the ackPayload concept
The complications of swapping roles can be completely avoided by using the ackPayload concept. The idea is that the slave puts data in the ackPayload buffer BEFORE it receives a message from the master and then the data in the ackPayload buffer is sent automatically as part of the normal acknowledgment process without your code needing to make the devices swap roles.

The ackPayload concept can only be used when the amount of data transferring from slave to master is 32 bytes or less.

Saltar al contenido principal

It also means that the data sent from the slave is always a step behind the data received by the slave. For example the ackPayoad cannot take account of the data received in the message for which it is the acknowledgment.

ackPayload example
In this example the slave will send a pair of numbers back to the master.

SimpleTxAckPayload.ino

```
// SimpleTxAckPayload - the master or the

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>


#define CE_PIN   9
#define CSN_PIN 10

const byte slaveAddress[5] = {'R','x','A',

RF24 radio(CE_PIN, CSN_PIN); // Create a R

char dataToSend[10] = "Message 0";
char txNum = '0';
int ackData[2] = {-1, -1}; // to hold the
bool newData = false;

unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 1000; //

//===============

void setup() {

    Serial.begin(9600);
    Serial.println(F("Source File /mnt/sdb
    Serial.println("SimpleTxAckPayload Sta

    radio.begin();
    radio.setDataRate( RF24_250KBPS );

    radio.enableAckPayload();
```

Saltar al contenido principal

```
    radio.setRetries(5,5); // delay, count
                                          //
    radio.openWritingPipe(slaveAddress);
}

//=============

void loop() {

    currentMillis = millis();
    if (currentMillis - prevMillis >= txIn
        send();
    }
    showData();
}

//================

void send() {

    bool rslt;
    rslt = radio.write( &dataToSend, sizeo
        // Always use sizeof() as it gives
        // For example if dataToSend was a

    Serial.print("Data Sent ");
    Serial.print(dataToSend);
    if (rslt) {
        if ( radio.isAckPayloadAvailable()
            radio.read(&ackData, sizeof(ac
            newData = true;
        }
        else {
            Serial.println("  Acknowledge
        }
        updateMessage();
    }
    else {
        Serial.println("  Tx failed");
    }

    prevMillis = millis();
 }
```

Saltar al contenido principal

```
//================

void showData() {
    if (newData == true) {
        Serial.print("  Acknowledge data "
        Serial.print(ackData[0]);
        Serial.print(", ");
        Serial.println(ackData[1]);
        Serial.println();
        newData = false;
    }
}

//================

void updateMessage() {
        // so you can see that new data is
    txNum += 1;
    if (txNum > '9') {
        txNum = '0';
    }
    dataToSend[8] = txNum;
}
```

SimpleRxAckPayload.ino

```
// SimpleRxAckPayload- the slave or the re

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define CE_PIN   9
#define CSN_PIN 10

const byte thisSlaveAddress[5] = {'R','x',

RF24 radio(CE_PIN, CSN_PIN);

char dataReceived[10]; // this must match
int ackData[2] = {109, -4000}; // the two
bool newData = false;

//==============
```

```
void setup() {

    Serial.begin(9600);

    Serial.println("SimpleRxAckPayload Sta
    radio.begin();
    radio.setDataRate( RF24_250KBPS );
    radio.openReadingPipe(1, thisSlaveAddr

    radio.enableAckPayload();

    radio.startListening();

    radio.writeAckPayload(1, &ackData, siz
}

//==========

void loop() {
    getData();
    showData();
}

//============

void getData() {
    if ( radio.available() ) {
        radio.read( &dataReceived, sizeof(
        updateReplyData();
        newData = true;
    }
}

//================

void showData() {
    if (newData == true) {
        Serial.print("Data received ");
        Serial.println(dataReceived);
        Serial.print(" ackPayload sent ");
        Serial.print(ackData[0]);
        Serial.print(", ");
        Serial.println(ackData[1]);
        newData = false;
    }
}
```

```
//================

void updateReplyData() {
    ackData[0] -= 1;
    ackData[1] -= 1;
    if (ackData[0] < 100) {
        ackData[0] = 109;
    }
    if (ackData[1] < -4009) {
        ackData[1] = -4000;
    }
    radio.writeAckPayload(1, &ackData, siz
}
```

Edit 05 Dec 2016 to correct the wrong pipe reference in the function updateReplyData(). Apologies for any confusion.

Edit 18 Mar 2017 - see Reply #17 for a version that works with multiple slaves

Edit 21 Feb 2019 to change setup() in RxAckPayload so the payload is uploaded after startListening()

Edit 20 Jan 2020 to increase delay in setRetries() to allow for longer ackPayloads. Details in section 7.4.2 of the Nordic nRF24L01+ datasheet

Continues in next Post

Two Way nrf communication

Working through problems with nrf24 and n...

Possibility control nrf24l01 by handy andrem...

Robin2 ago. '16 post #4

Two-way transmission by swapping roles
I have not found a need for this approach but I will illustrate it in case it is useful for someone else.

In this example the two nRF24s (master and slave) spend most of their time listening and only switch

Saltar al contenido principal

to talking for the minimum time needed to send a message. As with the ackPayload example the slave only sends a message in response to a message from the master.

MasterSwapRoles.ino

```
// MasterSwapRoles

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>


#define CE_PIN   9
#define CSN_PIN 10

const byte slaveAddress[5] = {'R','x','A',
const byte masterAddress[5] = {'T','X','a'


RF24 radio(CE_PIN, CSN_PIN); // Create a R

char dataToSend[10] = "Message 0";
char txNum = '0';
int dataReceived[2]; // to hold the data f
bool newData = false;

unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 1000; //

//============

void setup() {

    Serial.begin(9600);

    Serial.println("MasterSwapRoles Starti

    radio.begin();
    radio.setDataRate( RF24_250KBPS );

    radio.openWritingPipe(slaveAddress);
    radio.openReadingPipe(1, masterAddress
```

```
    radio.setRetries(3,5); // delay, count
    send(); // to get things started
    prevMillis = millis(); // set clock
}

//=============

void loop() {
    currentMillis = millis();
    if (currentMillis - prevMillis >= txIn
        send();
        prevMillis = millis();
    }
    getData();
    showData();
}

//====================

void send() {

        radio.stopListening();
            bool rslt;
            rslt = radio.write( &dataToSen
        radio.startListening();
        Serial.print("Data Sent ");
        Serial.print(dataToSend);
        if (rslt) {
            Serial.println("  Acknowledge
            updateMessage();
        }
        else {
            Serial.println("  Tx failed");
        }
}

//================

void getData() {

    if ( radio.available() ) {
        radio.read( &dataReceived, sizeof(
        newData = true;
    }
}
```

```
//================

void showData() {
    if (newData == true) {
        Serial.print("Data received ");
        Serial.print(dataReceived[0]);
        Serial.print(", ");
        Serial.println(dataReceived[1]);
        Serial.println();
        newData = false;
    }
}

//================

void updateMessage() {
        // so you can see that new data is
    txNum += 1;
    if (txNum > '9') {
        txNum = '0';
    }
    dataToSend[8] = txNum;
}
```

SlaveSwapRoles.ino

```
// SlaveSwapRoles

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>


#define CE_PIN   9
#define CSN_PIN 10

const byte slaveAddress[5] = {'R','x','A',
const byte masterAddress[5] = {'T','X','a'

RF24 radio(CE_PIN, CSN_PIN); // Create a R

char dataReceived[10]; // must match dataT
int replyData[2] = {109, -4000}; // the tw
bool newData = false;
```

Saltar al contenido principal

```
unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 1000; //


void setup() {

    Serial.begin(9600);

    Serial.println("SlaveSwapRoles Startin

    radio.begin();
    radio.setDataRate( RF24_250KBPS );

    radio.openWritingPipe(masterAddress);
    radio.openReadingPipe(1, slaveAddress)

    radio.setRetries(3,5); // delay, count
    radio.startListening();

}

//====================

void loop() {
    getData();
    showData();
    send();
}

//====================

void send() {
    if (newData == true) {
        radio.stopListening();
            bool rslt;
            rslt = radio.write( &replyData
        radio.startListening();

        Serial.print("Reply Sent ");
        Serial.print(replyData[0]);
        Serial.print(", ");
        Serial.println(replyData[1]);

        if (rslt) {
            Serial.println("Acknowledge Re
```

```
            updateReplyData();
        }
        else {
            Serial.println("Tx failed");
        }
        Serial.println();
        newData = false;
    }
}

//================

void getData() {

    if ( radio.available() ) {
        radio.read( &dataReceived, sizeof(
        newData = true;
    }
}

//================

void showData() {
    if (newData == true) {
        Serial.print("Data received ");
        Serial.println(dataReceived);
    }
}

//================

void updateReplyData() {
    replyData[0] -= 1;
    replyData[1] -= 1;
    if (replyData[0] < 100) {
        replyData[0] = 109;
    }
    if (replyData[1] < -4009) {
        replyData[1] = -4000;
    }
}
```

End of Tutorial

Saltar al contenido principal

...R

Edit 02 September 2019 to remove reference to broken link

Help - NRF24L01+_Multiple Transceivers / Pip...

Two Way nrf communication

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal

Saltar al contenido principal