**NRF24L01+**

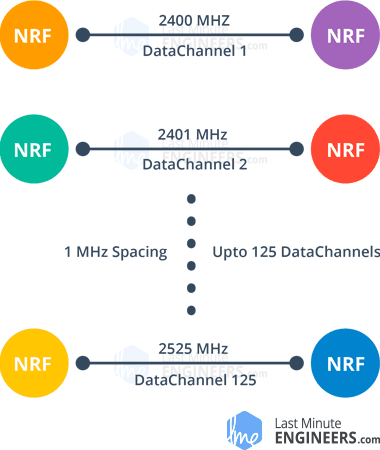
**Outline:**

1. Introduction.
   1. RF Channel Frequency.
   2. nRF24L01+ Multiceiver Network.
   3. Important Features of nRF24L01 IC.
2. nRF24L01 Transceiver Module.
   1. Components on nRF24L01 Transceiver Module.
   2. Pins Configuration of nRF24L01 Transceiver Module.
3. Interfacing nRF24L01 Transceiver Module with Arduino.
   1. nRF24L01 Transceiver Module Adapter.
4. Components Required.
5. Simple Arduino Wireless Communication using nRF24L01 Transceiver Module.
   1. Unidirectional Communication:
      1. Circuit Diagram.
      2. Connections.
      3. Code.
         1. Transmitter Code.
         2. Receiver Code.
         3. Code Explanation.
   2. Bidirectional Communication:
      1. Circuit Diagram.
      2. Code.
         1. Transmitter Code.
         2. Receiver Code.
      3. Code Explanation.
6. Troubleshoot.
   1. Power Supply Noise.
   2. External Power Supply.
7. Resources.
8. **Introduction:**

A single chip RF Transceiver IC developed by Nordic Semiconductor. It operates in the license-free 2.4GHz ISM band (ISM – Industrial, Scientific and Medical) with support for data rates of 250kbps, 1Mbps and 2Mbps.

* 1. **RF Channel Frequency:**

nRF24L01+ transceiver module transmits and receives data on a certain frequency called Channel. Also in order for two or more transceiver modules to communicate with each other, they need to be on the same channel. This channel could be any frequency in the 2.4 GHz ISM band (2.400 to 2.525 GHz).

Each channel occupies a bandwidth of less than 1MHz. This gives us 125 possible channels with 1MHz spacing. So, the module can use 125 different channels which give a possibility to have a network of 125 independently working modems in one place.

250kbps and 1Mbps data rates channels bandwidth = 1MHz.

2Mbps data rate channel bandwidth should be 2MHz or more to ensure non-overlapping.

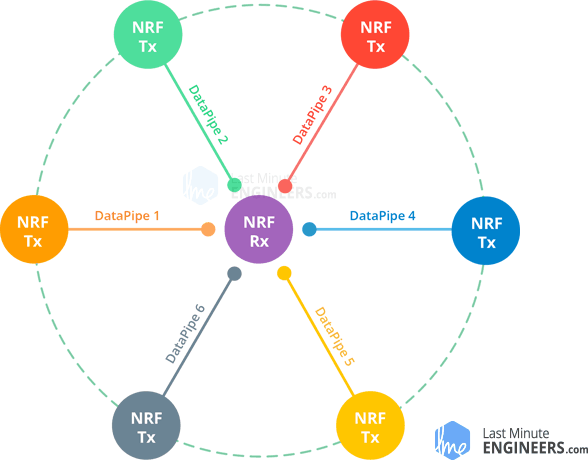
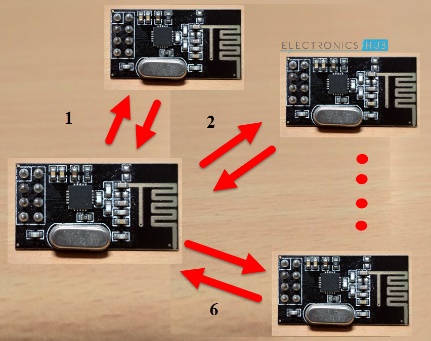
RF channel frequency of your selected channel is set according to the following formula:

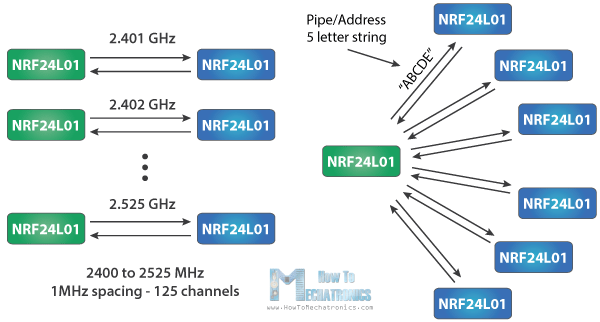
Freq(Selected) = 2400 + CH(Selected)

Example: if you select channel 108 for data transmission, RF channel frequency of channel = 2508MHz 🡺 (2400 + 108)

* 1. **nRF24L01+ Multiceiver Network:**

nRF24L01+ IC has MultiCeiver(Multiple Transmitters Single Receiver) feature enables each RF Channel to be logically divided into 6 unique addressed parallel data channels called Data Pipes. So that each module can communicate with 6 other modules in the same RF Channel



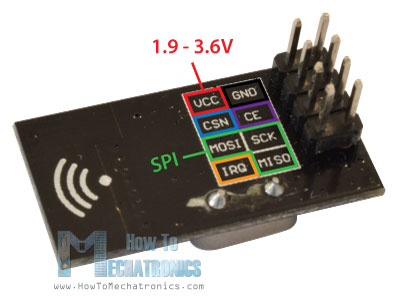


Imagine the primary receiver acting as a hub receiver:

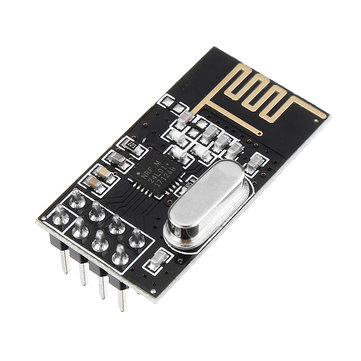
1. collects information from 6 different transmitter nodes simultaneously.
2. can stop listening any time and acts as a transmitter. But this can only be done one pipe/node(send to one node) at a time.

To design a Wireless Communication system using nRF24L01 Module, all you need is a Microcontroller, which is interfaced through Serial Peripheral Interface (SPI).

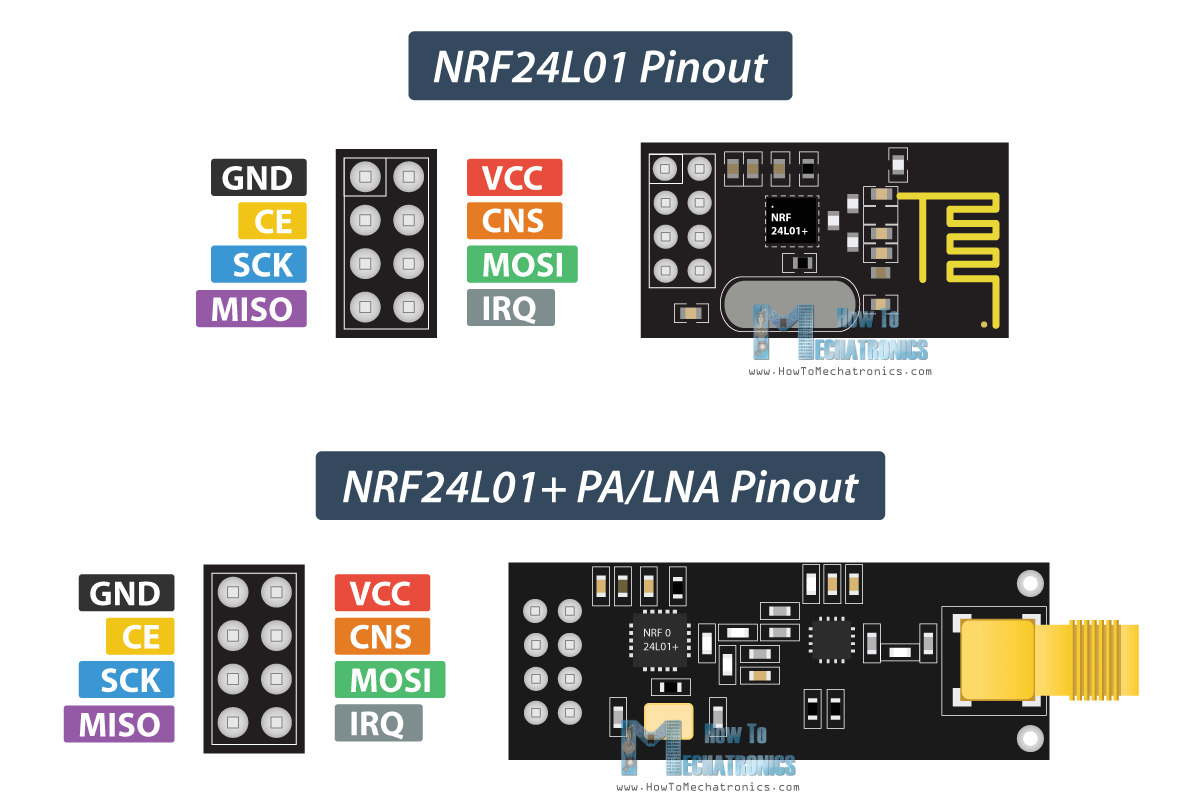
* 1. **Important Features of nRF24L01 IC:**

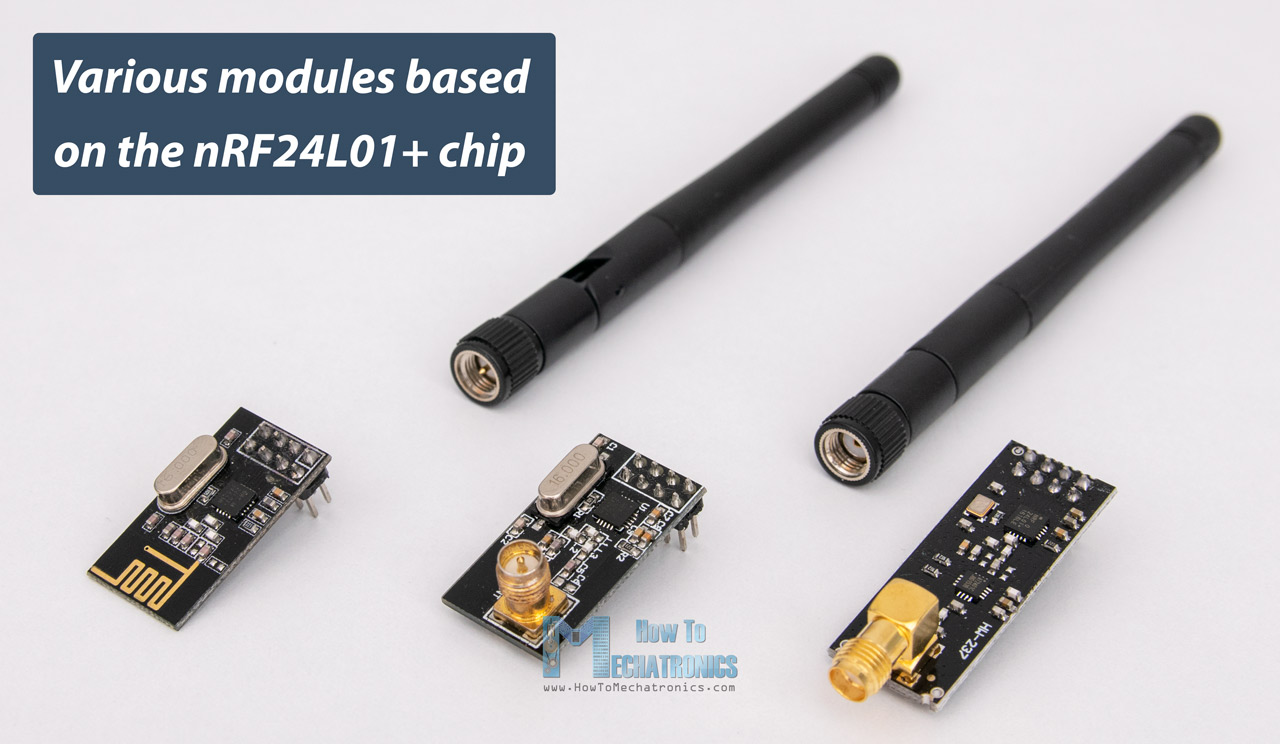
1. Ultra-low power operation (26μA Standby-I mode, 900nA power down mode).
2. 12mA during transmission.
3. SPI Interface with Microcontroller.
4. Integrated RF Transmitter, Receiver and Synthesizer.
5. Operating voltage is 1.9V – 3.6V (3.3v).
6. Other 6 data pins tolerate 5V logic, so we can easily connect it to an Arduino without using any logic level converters.

nRF24L01 Transceiver Module.

1. **nRF24L01 Transceiver Module:**
   1. **Components on nRF24L01 Transceiver Module:**

nRF24L01+ Transceiver IC, Antenna Trace (on the PCB), Pins for communication and power and a few passive components.

* 1. **Pins Configuration of nRF24L01 Transceiver Module:**
     1. VCC : 3.3V must be given.
     2. GND : Ground.
     3. SCK : SPI Clock Pin.
     4. MOSI : SPI Slave Data Input Pin.
     5. MISO : SPI Slave Data Output Pin.
     6. IRQ : Active LOW Interrupt Pin.
     7. CE : Chip Enable Pin.
     8. CSN : SPI Chip Select Pin.



1. NRF24L01: uses on-board antenna. This allows for a more compact version of the breakout. However, the smaller antenna also means a lower transmission range; over a distance of 100 meters. Of course, that is outdoors in an open space. Your range indoors, especially through walls, will be slightly weakened.
2. NRF24L01+PA/LNA: comes with a SMA connector and a duck-antenna but that’s not the real difference.

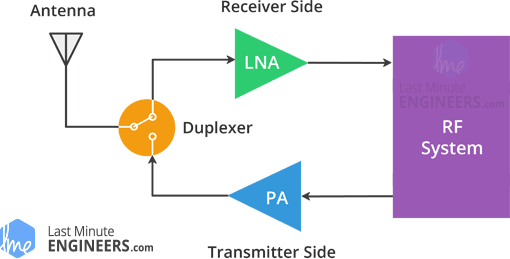
The real difference is that it comes with RFX2401C chip which integrates the PA, LNA, and transmit-receive switching circuitry.

This range extender chip along with a duck-antenna helps the module achieve a significantly larger transmission range about 1000m.

What is PA/LNA?

PA (Power Amplifier): merely boosts the power of the signal being transmitted from the nRF24L01+ chip.

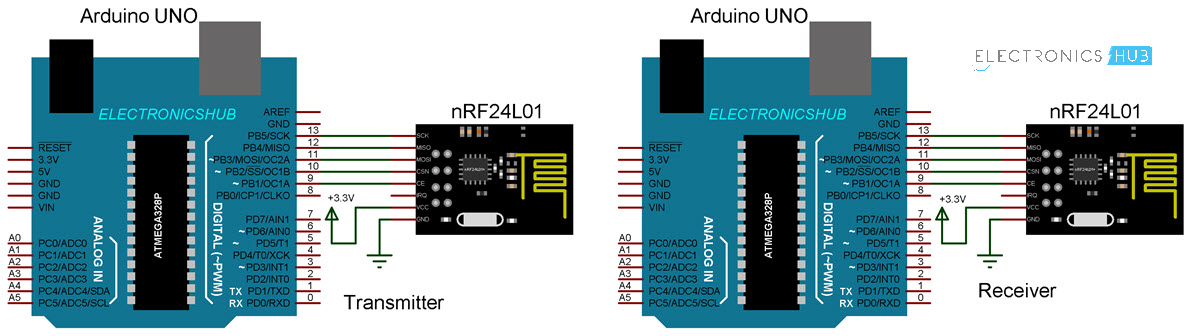
LNA (Low-Noise Amplifier): take the extremely weak and uncertain signal from the antenna (usually on the order of microvolts or under -100 dBm) and amplify it to a more useful level (usually about 0.5 to 1V)

LNA of the receive path and PA of the transmit path connect to the antenna via a duplexer, which separates the two signals and prevents the relatively powerful PA output from overloading the sensitive LNA input.

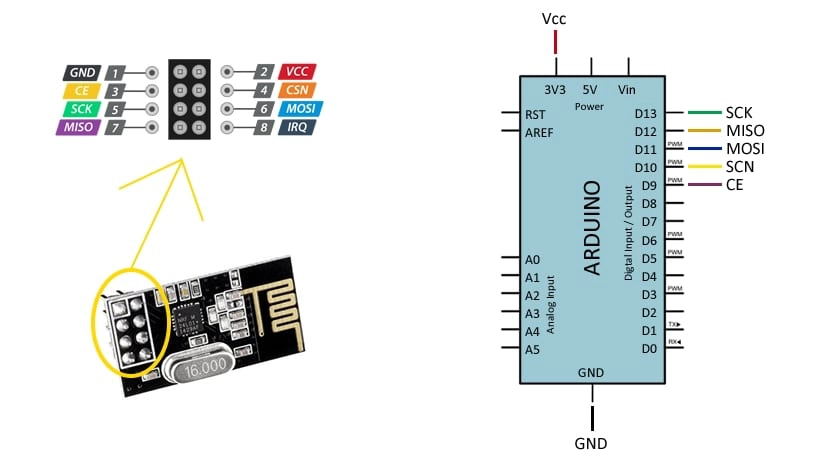
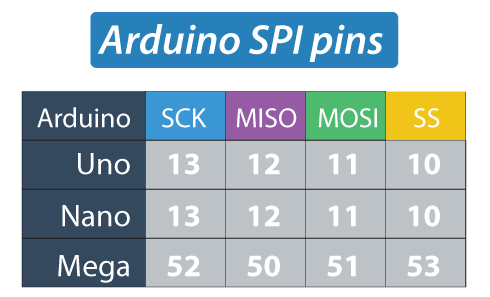
1. **Interfacing nRF24L01 Transceiver Module with Arduino:**

nRF24L01 Transceiver Module communicates over SPI Interface. So, to interface with Arduino, you need to use the SPI Pins of the Arduino board.

1. **Simple Arduino Wireless Communication using nRF24L01 Transceiver Module:**
   1. **Unidirectional Communication:**
      1. **Circuit Diagram:**



* + 1. **Connections:**



**Note:**

MOSI, MISO, and SCK pins must be digital pins 11, 12, and 13 respectively.

CE and CSN pins can be any two digital pins! They are used to:

1. Setting the board in standby or active mode.
2. Switching between transmit and receive command mode.
   * 1. **Code:**
        1. **Transmitter Code:**

|  |
| --- |
| #include <SPI.h> |
| #include <nRF24L01.h> |
| #include <RF24.h> |
| RF24 radio(9, 10); // CE, CSN Pins |
| const uint64\_t address = 0xF0F0F0F0E1LL; |
|  |
| void setup() |
| { |
| radio.begin(); |
| radio.openWritingPipe(address); |
| radio.setPALevel(RF24\_PA\_MIN); |
| radio.stopListening(); |
| } |
| void loop() |
| { |
| const char sendData[] = "This is a test!"; |
| radio.write(&sendData, sizeof(sendData)); |
| delay(500); |
| } |

* + - 1. **Receiver Code:**

|  |
| --- |
| #include <SPI.h>  #include <nRF24L01.h> |
| #include <RF24.h> |
| RF24 radio(9, 10); // CE, CSN Pins |
| const uint64\_t address = 0xF0F0F0F0E1LL; |
|  |
|  |
| void setup() |
| { |
| Serial.begin(9600); |
| radio.begin(); |
| radio.openReadingPipe(1, address); |
| radio.setPALevel(RF24\_PA\_MIN); |
| radio.startListening(); |
| } |
| void loop() |
| { |
| if (radio.available()) { |
| char recvData[32] = ""; |
| radio.read(&recvData, sizeof(recvData)); |
| Serial.println(recvData); |
| } |
| } |

* + - 1. **Code Explanation:**

1. We need to include the basic SPI and the newly installed RF24 libraries and create an RF24 object. The two arguments here are the CSN and CE pins.

RF24 radio(9, 10); // CE, CSN Pins

1. We need to create a byte array represents the address, or the so called pipe through which the two modules will communicate.

const uint64\_t address = 0xF0F0F0F0E1LL;

OR

const byte address[6] = "00001";

We can change the value of this address to any 5 letter string and this enables to choose to which receiver we will talk, so in our case we will have the same address at both the receiver and the transmitter.

1. To initialize the radio object and transmit using the radio.openWritingPipe(),, we set the address of the receiver has the 5 letter string we previously set.

radio.openWritingPipe(address);

1. At the receiver, using the radio.setReadingPipe() function we set the same address which enables the communication between the two modules.

radio.openReadingPipe(1, address);

1. radio.setPALevel() sets the Power Amplifier level in case of PA/LNA module.
   1. If modules are very close to each other, use:

radio.setPALevel(RF24\_PA\_MIN);

OR

radio.setPALevel(RF24\_PA\_LOW);

* 1. If modules are far from each other, use:
     1. Bypass capacitor (104) across GND and 3.3V of the modules.
     2. radio.setPALevel(RF24\_PA\_HIGH);

1. radio.stopListening() sets module as transmitter. On the other side, radio.startListening() sets the module as receiver.

radio.stopListening(); // at the Transmitter

radio.startListening(); // at the Receiver

1. In loop section:
   1. Transmitter: create an array of characters to which we assign the message text. Using the radio.write() function we will send that message to the receiver.

radio.write() takes:

1. The address of the variable to be sent.
2. Number of bytes to be sent.

|  |
| --- |
| void loop() |
| { |
| const char sendData[] = "This is a test!"; |
| radio.write(&sendData, sizeof(sendData)); |
| delay(500); |
| } |

we will add 0.5 second delay.

* 1. Receiver:

radio.available() check whether there is data to be received. If that’s true, first we create an array of 32 elements, called “text”, in which we will save the incoming data.

|  |
| --- |
| void loop() |
| { |
| if (radio.available()) { |
| char recvData[32] = ""; |
| radio.read(&recvData, sizeof(recvData)); |
| Serial.println(recvData); |
| } |
| } |

In RF24 library documentation, there are many parameters that can be set. Some key parameters are:

1. Channel: the specific frequency channel that communication will occur on (frequencies are mapped to integers between 0 and 125
2. Reading pipe: a unique 24, 32, or 40-bit address the module reads data from.
3. Writing pipe: a unique address to which the module writes data
4. Power Amplifier (PA) level: the PA level sets the power draw of the chip and thereby the transmission power. For the purposes of this tutorial (use with the Arduino) we will use the minimum power setting.

In general, when using the RF module with an Arduino board, it is probably a good idea to keep the PA level as low as possible to reduce the current draw on the Arduino’s regulated power supply.

At higher levels (further distances), 104 ceramic bypass capacitor at module i/p.

A good programming practice for storing the writing and reading pipe addresses is to put the two values inside an array (byte array named “addresses.”)

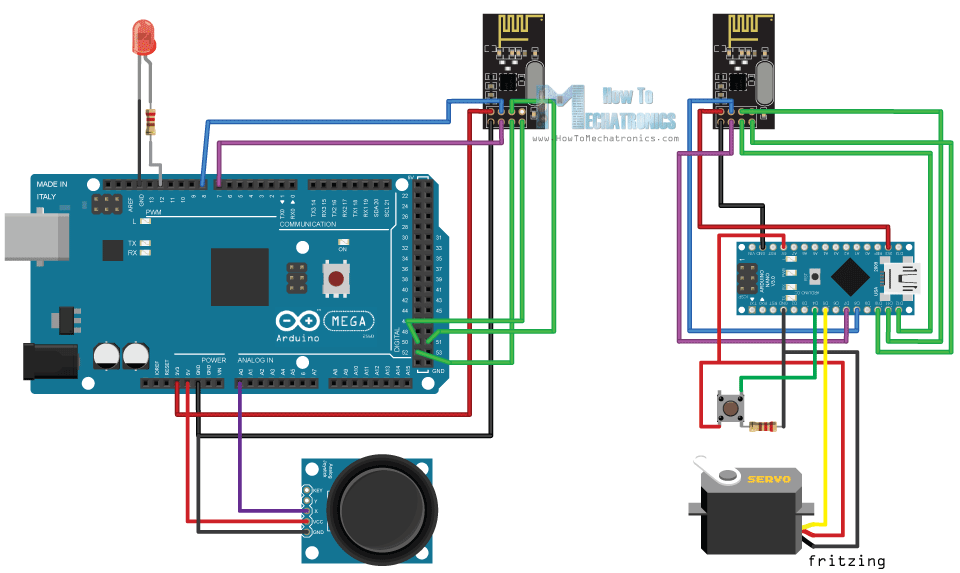
Note that the openReadingPipe() method must be passed an additional integer parameter that describes which reading pipe is being initialized. This is because the RF module can receive from up to 6 pipes at a time!

Listening is done by calling the RF24::startListening() method.

Note: the reading pipe must be initialized before start listening for data (openReadingPipe() method must be called before the startListening() method!)

Similarly, stopListening() method must be called before writing.

if data is available over RF connection, available()returns true, and data can be read.

* 1. **Bidirectional Communication:**
     1. **Circuit Diagram:**
     2. **Code:**
        1. **Transmitter Code:**

1. /\*
2. \* Arduino Wireless Communication Tutorial
3. \* Example 2 - Transmitter Code
4. \*
5. \* by Dejan Nedelkovski, www.HowToMechatronics.com
6. \*
7. \* Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
8. \*/
9. #include <SPI.h>
10. #include <nRF24L01.h>
11. #include <RF24.h>
12. #define led 12
13. RF24 radio(7, 8); // CE, CSN
14. const byte addresses[][6] = {"00001", "00002"};
15. boolean buttonState = 0;
16. void setup() {
17. pinMode(12, OUTPUT);
18. radio.begin();
19. radio.openWritingPipe(addresses[1]); // 00002
20. radio.openReadingPipe(1, addresses[0]); // 00001
21. radio.setPALevel(RF24\_PA\_MIN);
22. }
23. void loop() {
24. delay(5);
25. radio.stopListening();
26. int potValue = analogRead(A0);
27. int angleValue = map(potValue, 0, 1023, 0, 180);
28. radio.write(&angleValue, sizeof(angleValue));
29. delay(5);
30. radio.startListening();
31. while (!radio.available());
32. radio.read(&buttonState, sizeof(buttonState));
33. if (buttonState == HIGH) {
34. digitalWrite(led, HIGH);
35. }
36. else {
37. digitalWrite(led, LOW);
38. }
39. }
    * + 1. **Receiver Code:**
40. /\*
41. \* Arduino Wireless Communication Tutorial
42. \* Example 2 - Receiver Code
43. \*
44. \* by Dejan Nedelkovski, www.HowToMechatronics.com
45. \*
46. \* Library: TMRh20/RF24, https://github.com/tmrh20/RF24/
47. \*/
48. #include <SPI.h>
49. #include <nRF24L01.h>
50. #include <RF24.h>
51. #include <Servo.h>
52. #define button 4
53. RF24 radio(7, 8); // CE, CSN
54. const byte addresses[][6] = {"00001", "00002"};
55. Servo myServo;
56. boolean buttonState = 0;
57. void setup() {
58. pinMode(button, INPUT);
59. myServo.attach(5);
60. radio.begin();
61. radio.openWritingPipe(addresses[0]); // 00001
62. radio.openReadingPipe(1, addresses[1]); // 00002
63. radio.setPALevel(RF24\_PA\_MIN);
64. }
65. void loop() {
66. delay(5);
67. radio.startListening();
68. if ( radio.available()) {
69. while (radio.available()) {
70. int angleV = 0;
71. radio.read(&angleV, sizeof(angleV));
72. myServo.write(angleV);
73. }
74. delay(5);
75. radio.stopListening();
76. buttonState = digitalRead(button);
77. radio.write(&buttonState, sizeof(buttonState));
78. }
79. }
    * + 1. **Code Explanation:**

What’s different here from the previous example is that:

* 1. Ccreate two pipes or addresses for the bi-directional communication.

const byte addresses[][6] = {"00001", "00002"};

* 1. In the setup section, define both pipes, and note that the writing address at the first Arduino needs to be the reading address at the second Arduino, and vice versa, the reading address at the first Arduino needs to be the writing address at the second Arduino.
     1. Transmitter:

radio.openWritingPipe(addresses[1]); // 00001

radio.openReadingPipe(1, addresses[0]); // 00002

* + 1. Receiver:

radio.openWritingPipe(addresses[0]); // 00002

radio.openReadingPipe(1, addresses[1]); // 00001

* 1. In the loop section using the radio.stopListening() function we set the first Arduino as transmitter, read and map the value of Joystick from 0 to 180, and using the radio.write() function send the data to the receiver.

radio.stopListening();

int potValue = analogRead(A0);

int angleValue = map(potValue, 0, 1023, 0, 180);

radio.write(&angleValue, sizeof(angleValue));

* 1. On the other side, using the radio.startListening() function we set the second Arduino as receiver and we check whether there is available data. While there is data available we will read it, save it to the “angleV” variable and then use that value to rotate the servo motor.

radio.startListening();

if ( radio.available()) {

while (radio.available()) {

int angleV = 0;

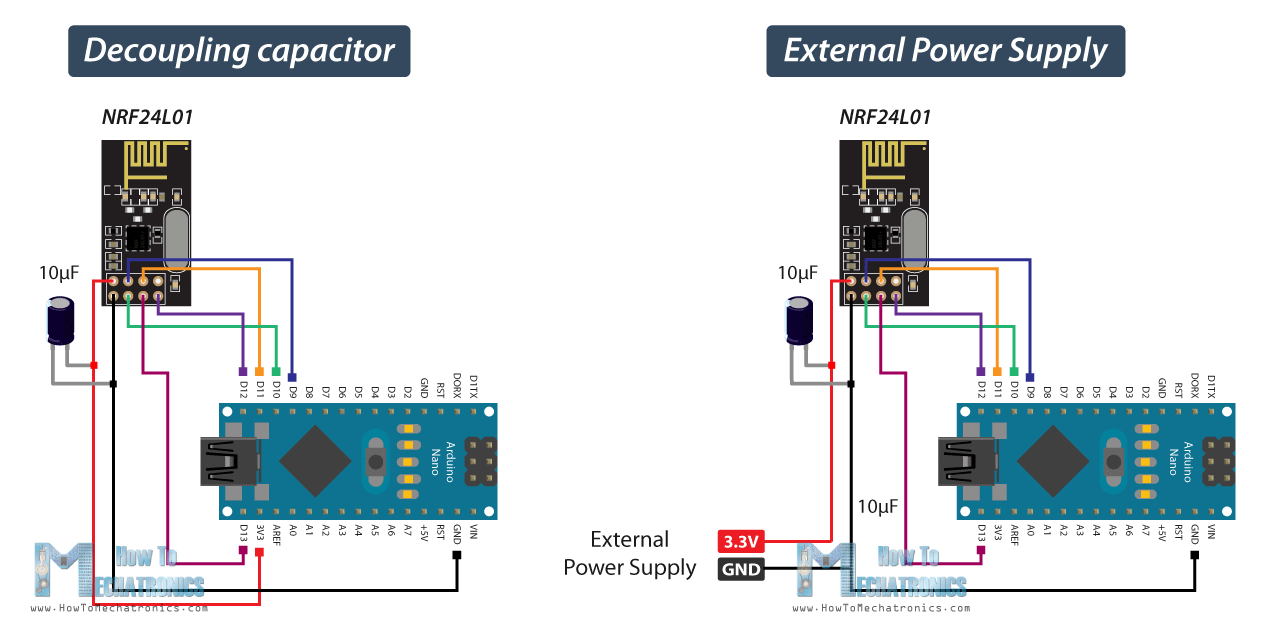
radio.read(&angleV, sizeof(angleV));

myServo.write(angleV);

}

* 1. Next, at the transmitter, we set the first Arduino as receiver and with an empty “while” loop we wait for the second Arduino the send data, and that’s the data for the state of the push button whether is pressed or not. If the button is pressed the LED will light up. So, these processes constantly repeats and both Arduino boards are constantly sending and receiving data.

1. **Troubleshoot:**
   1. **Power Supply Noise:**

It is one of the most common issues people experience when trying to make successful communication with the NRF24L01 modules. Generally, RF circuits or radio frequency signals are sensitive to power supply noise. Therefore, it’s always a good idea to include a decoupling capacitor across the power supply line. The capacitor can be anything from 10uF to 100uF.

* 1. **External Power Supply:**

The 3.3V pin of the Arduino boards cannot always supply enough power to the NRF24L01 module. So, powering the module with an external power source is also a good idea.

1. **Resources:**
   1. ElectronicsHub, nRF24L01 Transceiver Module: <https://www.electronicshub.org/nrf24l01-transceiver-module/>
   2. HowToMechatronics, nRF24L01 – How It Works, Arduino Interface, Code, Schematic: <https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/>
   3. LastMinuteEngineering, How nRF24L01+ Wireless Module Works & Interface with Arduino: <https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/>