Description

NRG240L01 RF Board (A)

The NRF24L01 RF Board (A) is an accessory board offers wireless 2.4G solution with NRF24L01 on board. It uses SPI interface for communication. The board is ideal for applications such as wireless data transmission, multicast, and frequency-hopping communication.

Special Note

- *The NRF24L01 RF Board (A) integrates only ONE NRF24L01, while at least TWO NRF24L01 are required to establish communication with each other.*
- *Operating Voltage: 1.9V~3.6V low voltage power supply only.*
- *Communication distance: almost no loss within 50 meters. (Software settings: 250kbps, 0dBm, low noise amplifier gain; working environment: open area)*

NRF24L01 RF Board (B)

The NRF24L01 RF Board (B) is an accessory board offers wireless 2.4G solution with NRF24L01 on board. It uses SPI interface for communication. The board is ideal for applications such as wireless data transmission, multicast, and frequency-hopping communication.

Features:

- Its pin definition is compatible to the SPI port of STM32 Open series development board.
- Resisters and capacitors are packaged in RC0402. Has better quality than type A

Special Note

- *The NRF24L01 RF Board (B) integrates only ONE NRF24L01, while at least TWO NRF240L01 are required to establish communication with each other.*
- *Operating Voltage: 1.9V ~ 3.6V low voltage power supply only.*
- *Communication distance: almost no loss within 55 meters. (Software settings 250kbps, 0dBm, low noise amplifier gain; working environment: open area)*

NRF24L01 RF Board (C)

The NRF24L01 RF Board (C) is an accessory board offers wireless 2.4G solution with NRF24L01 on board. It uses SPI interface for communication. The board is ideal for applications such as wireless data transmission, multicast, and frequency-hopping communication.

Features:

- Supports 2Mbps data rate.
- Supports 6 node networks.

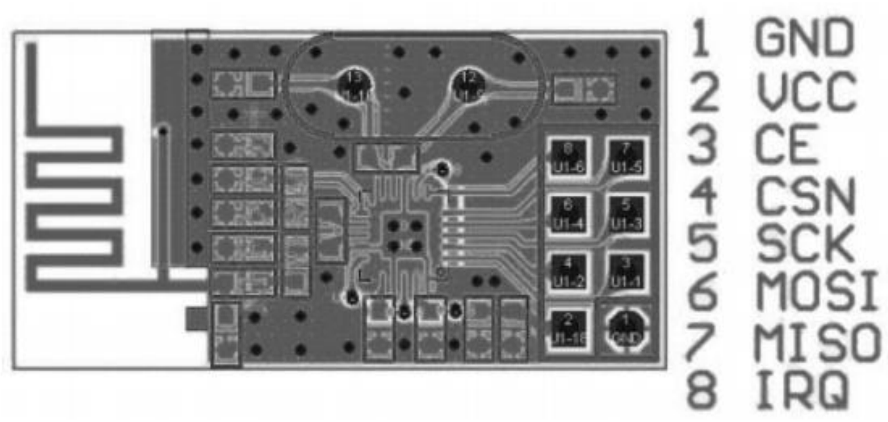
- Supports 125 radio channels
- Has PA amplification for ultra-long-distance transmission.
- Has LNA low noise amplifier for ultra-long distance transmission.
- Can work with external high sensitivity antenna. (Optional)
- Power output can be set by software

Special Note:

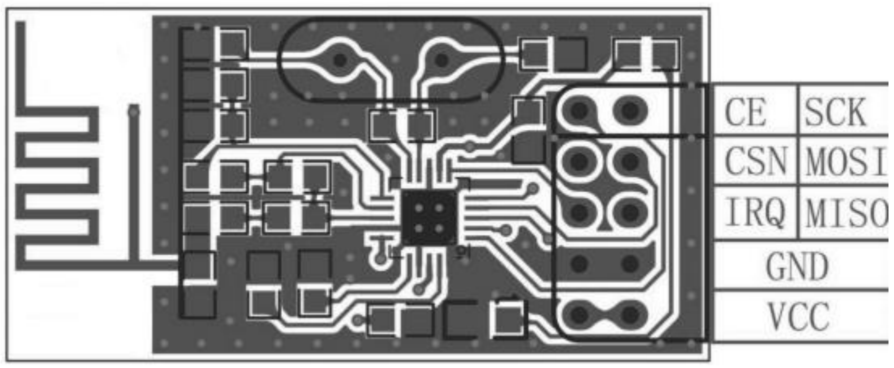
- *The NRF24L01 RF Board (C) integrates only ONE NRF24L01, while at least TWO NRF24L01 are required to establish communication with each other.*
- *Operating Voltage: 3V~3.6V power supply only.*
- *Communication distance: in open area, with 2DB external antenna. Almost no loss within 1000 meters. (Software settings: 250kbps, 0dBm, low noise amplifier gain). Almost no loss within 700 meters. (Software settings: 1Mbps, 0dBm, low noise amplifier gain). Almost no loss within 500 meters. (Software settings: 2Mbps, 0dBm, low noise amplifier gain).*

External antenna is NOT included in the price. You should buy one separately for long-distance communication. Recommended antenna: RP-SMA 2.4G 2DB Antenna

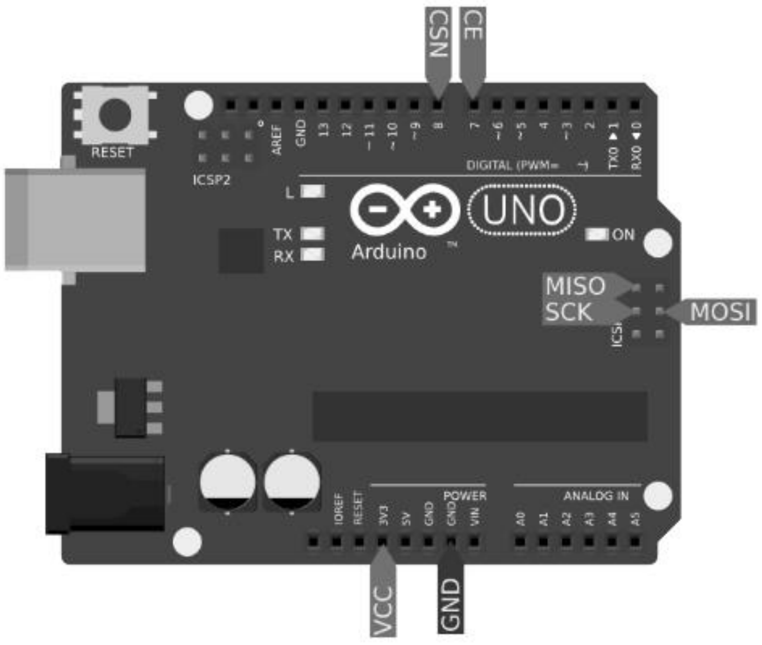
Pin Definition:



NRF24L01 RF Board (A & C) Pin Definition



NRF24L01 RF Board (B) Pin Definition



Schematic is generally universal and fits all the Arduino's: UNO, DUE, MEGA, Leonardo, Yun etc. (Arduino 1.0 (R3) standard, but also with older boards)

SPI signals are in the ICSP connector. For connecting we suggest using female/female jumper wires (type FF). The rest of the signals can be connected using a female/male jumper wires (type FM).

If required can be purchased from: https://www.phippselectronics.com

Connect power pins from NRF24L01 to the Arduino as shown below:

NRF24L01	**ARDUINO**
VCC	3.3V
GND	GND

CE and CSN pins can be connected to any digital pins. Then in RF24 library, you can specify which pins you used. I chose pins 7 and 8 because I will use them in the examples.

On Arduino UNO boards SPI pins are connected with some digital pins. While using modem you must remember that these digital pins won't be available.

- MOSI is connected to the digital pin 11
- MISO is connected to the digital pin 12
- SCK is connected to the digital pin 13
- SS (not used, but also blocks) is connected to the digital pin 10

The Arduino MEG 1280 and 2560 have a similar configuration.

- MOSI is connected to the digital pin 51

- MISO is connected to the digital pin 50
- SCK is connected to the digital pin 52
- SS is connected to the digital pin 53

On the Arduino DUE, Yun and Leonardo SPI pins are on ICSP connector, and are independent of the digital pins.

Programming

Having the module connected, we need to program it. First program you probably know, we'll make traditional "Hello World".

We will make one device (with the modem), will send the string to the other device. The second device will send the received string to a stationary computer and they will display it in the Arduino Serial Port Monitor.

In this project we used RF24 library, which can be found on Github: RF24 library on Github. You only need to click on "Download ZIP" button and it'll start downloading all necessary things. You can install the library in Arduino IDE using Sketch-> Import library-> Add library. Another way is to extract the zip file to your Arduino home directory: Arduino/libraries on Linux or Documents/ Arduino/libraries in Windows.

Transmitter program will look like:

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

RF24 radio(7, 8);

const byte rxAddr[6] = "00001";

void setup()
{
  radio.begin();
  radio.setRetries(15, 15);
  radio.openWritingPipe(rxAddr);

  radio.stopListening();
}

void loop()
{
  const char text[] = "Hello World";
  radio.write(&text, sizeof(text));

  delay(1000);
}
```

At the beginning of the sketch we inform the program that we'll use libraries.

- SPI.h – to handle the communication interface with the modem
- nRF24L01.h – to handle this particular modem driver
- RF24.h – the library which helps us to control the radio modem

Next, we need to create an object called "**radio**"

```
RF24 radio(7, 8);
```

This object represents a modem connected to the Arduino. Arguments 7 and 8 are a digital pin numbers to which signals CE and CSN are connected. If you have connected them to other pins can change this arguments. Then I create a global array called "**rxAddr**".

```
const byte rxAddr[6] = "00001";
```

In this array we wrote the address of the modem that will receive data from Arduino. Address has value "00001", but if you want you can change it to any other 5-letter string. The address is necessary if you have a few modems in the network, thanks to the address, you can choose a particular modem to which you are sending the data. In the "setup" function we call the method "**radio.begin ();**" . It activates the modem.

Next we call "**radio.setRetires(15, 15);**" function. It shows how many times the modem will retry to the send data in case of not receiving by another modem. The first argument sets how often modem will retry. It is a multiple of 250 microseconds. 15 * 250 = 3750. So, if the recipient does not receive data, modem will try to send them every 3.75 milliseconds. Second argument is the number of attempts. So in our example, modem will try to send 15 times before it will stop and finds that the receiver is out of range, or is turned off.

The method of "**radio.openWritingPipe (rxAddr);**" sets the address of the receiver to which the program will send data. Its argument is an array previously made with the receiver address.

The last method in the "setup" function is "**radio.stopListening ();**". It switch the modem to data transmission mode.

In the "loop" function, we start with creating a string that we want to send using modem.

```
const char text[] = "Hello World";
```

It's an array of characters/letters to which we assigned a "Hello World" text. Then, using the method of "**radio.write (& text, sizeof (text));**" we send text through the radio to the modem (the address of the modem was set up earlier using "**openWritingPipe**"). First argument is an indication of the variable that stores the data to send. That's why we used "**&**" before the variable name, so we can make an indicator from this variable. The second argument is the number of bytes that the radio will take from a variable to be sent. Here we used the function "**sizeof ()**", which automatically calculates the number of bytes in a "**text**" string.

Through this method, you can send up to 32 bytes at one time. Because that is the maximum size of a single packet data modem. If you need confirmation that the receiver received data, the method "**radio.write**" returns a "**bool**" value. If it returns "**true**" , the data reached the receiver. If it returns "**false**" this data has not been received.

The "**radio.write**" method blocks the program until it receives the acknowledgment or until you run out of all attempts to transmit established methods set in "**radio.setRetires**".

The last part of the "loop" function is "**delay (1000);**". It blocks the program for 1000 milliseconds, or one second. It makes the program will sent "Hello World" every second to the receiver.

The Receiver

The program of the receiver in the second modem device will look like this:

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

RF24 radio(7, 8);

const byte rxAddr[6] = "00001";

void setup()
{
  while (!Serial);
  Serial.begin(9600);

  radio.begin();
  radio.openReadingPipe(0, rxAddr);

  radio.startListening();
}

void loop()
{
  if (radio.available())
  {
    char text[32] = {0};
    radio.read(&text, sizeof(text));

    Serial.println(text);
  }
}
```

The program looks quite similar to the program of the transmitter. First we selected libraries which will be used and then we creates a “radio” object with selected control pins. In the next line you can see a table with the address of the receiver – the same as in the transmitter. At the beginning of the “setup” function we set the object “Serial” for communication Arduino with the computer.

```
while (! Serial);
```

This part is waiting for the Arduino USB port switches to serial COM port when You connect USB cable. That is true for Arduino with ATmega32u4 – like Leonardo, for ATmega328 based boards, which have separate chip for USB/Serial communication Serial is available always. The method of “**Serial.begin (9600);**” sets the baud rate with the computer via USB / COM.

The next part of the function is to set the nRF24L01 modem. Like before we used “**radio.begin ();**” method. The next line of the program is “**radio.openReadinPipe (0, rxAddr);**“, which determines the address of our modem which receives data. The first argument is the number of the stream. You can create up to 6 streams that responds to different addresses. We created only address for the stream number 0. The second argument is the address to which the stream will react to collect the data. In this example we set the address assigned to a “**rxAddr**” array.

The next step is to enable receiving data via modem using the method “**radio.startListening ();**“. From that moment the modem waits for data sent to the specified address. In the “loop” function, program performs the following operations. First checks whether any data have arrived at the address of the modem using the method “**radio.available ();**“. This method returns a “true” value if we received some data, or “false” if no data.

```
char text[32] = {0};
```

If the data was received, then it creates a 32-element “char” type array called “text” and filled with zeros (later the program will fill it with the received data). To read the data we use the method “**radio.read (& text, sizeof (text));**“. The first argument is an indicator of the variable to which you want to save the data received by the modem. To present a variable as an indicator we applied the “&” character in front of its name. The second argument is the amount of data to be stored in a variable. Here, we again have used the “**sizeof ()**” function, which automatically calculates the size of the “text” array.

When data is received, it sends it to the “**Serial.println (text);**” method. Then the received text is being sent to a computer, where you can see it in the “Serial Port Monitor” using the Arduino IDE. If you did everything ok and there are no mistakes in connections, you should see the same values in your Serial Port Monitor.