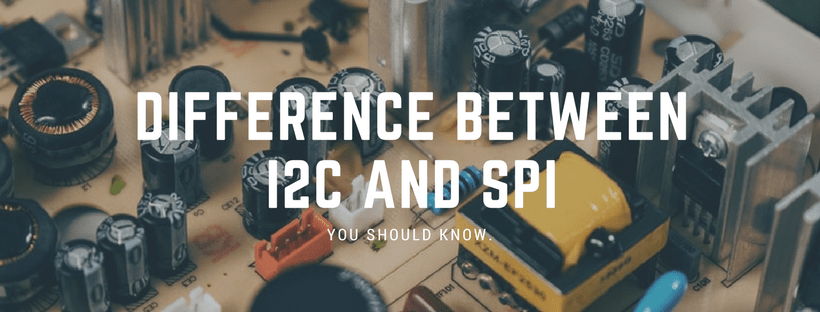
# Difference between I2C and SPI ( I2C vs SPI ), you should know.

BY [**AMLENDRA**](https://aticleworld.com/author/pritosh/)ON [**MARCH 16, 2018**](https://aticleworld.com/difference-between-i2c-and-spi/)

[**21**](https://aticleworld.com/difference-between-i2c-and-spi/#comments)



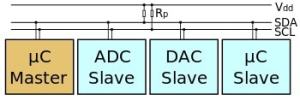
There is a lot of serial communication protocol but in which I2C and SPI are very famous, In this article, I will discuss the difference between I2C and SPI ( I2C vs SPI ). I2C and SPI both are bus protocol to allow the user for short-distance, serial data transfer. I2C is two-wire communication made by Philips (Nowadays NXP) and SPI is made by Motorola. Both protocols are commonly used in electronic devices like smartphones, TV, and laptops to control peripherals like power management chips, memory devices, input devices, etc.

**What is I2C?**

I2C is a serial communication protocol. It provides good support to the slow devices, for example, EEPROM, ADC, and RTC etc.I2c are not only used with the single board but also used with the other external components which have connected with boards through the cables.

I2C is basically a two-wire communication protocol. It uses only two-wire for communication. In which one wire is used for the data (SDA) and other wire is used for the clock (SCL).

In I2C, both buses are bidirectional, which means the master able to send and receive the data from the slave. The clock bus is controlled by the master but in some situations slave is also able to suppress the clock signal, but we will discuss it later.



Additionally, an I2C bus is used in the various control architecture, for example, SMBus (System Management Bus), PMBus (Power Management Bus), IPMI (Intelligent Platform Management Interface), etc.

**Read the article to know I2C protocol in detail:**[**Understanding of I2C Protocol.**](https://aticleworld.com/i2c-bus-protocol-and-interface/)

**Advantages of I2C communication protocol**

There is a lot of advantage of I2C protocol which makes the user helpless to use the I2C protocol in many applications.

* It is the synchronous communication protocol, so no need of precise oscillators for the master and slave.
* It requires only two-wire, one wire for the data (SDA), and other wire for the clock (SCL).
* It provides the flexibility to the user to select the transmission rate as per the requirements.
* In I2C Bus, each device on the bus is independently addressable.
* It follows the master and slave relationships.
* It has the capability to handle multiple masters and multiple slaves on the I2C Bus.
* I2C has some important features like arbitration, clock synchronization, and clock stretching.
* I2C provides ACK/NACK (acknowledgment/ Not-acknowledgement) features that provide help in error handling.

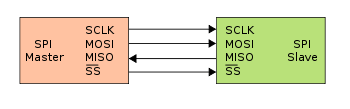
**Some important limitation of I2C communication protocol**

An I2C protocol has a lot of advantage but beside it, I2C has a few limitations.

* It consumes more power than other serial communication busses due to open-drain topology.
* It is good only for a short distance.
* I2C protocol has some limitation for the number of slaves, the number of the slave depends on the capacitance of the I2C bus.
* It only provides a few limited communication speed like 100 kbit/s,400 kbit/s, etc.
* In I2c, devices can set their communication speed, slower operational devices can delay the operation of faster speed devices.

**What is SPI?**

The serial peripheral interface is four wire-based full-duplex communication protocol these wire generally known as MOSI (master out slave in), MISO (master in slave out), SCL (a serial clock which produces by the master) and SS (slave select line which use to select specific slave during the communication).



SPI follows the master and slave architecture and communication is always started by the master. Like I2C it is also a synchronous communication protocol because the clock is shared by the master and slave.

SPI is supported only multi-slave does not support multi-master and slaves are selected by the slave select signal. In SPI during the communication data is shifted out from the master and shifted into the slave vice- versa through the shift register.

**Advantages of SPI communication protocol**

* There is no start and stop bits, so the data can be streamed continuously without interruption.
* It supports full-duplex.
* No need for precision oscillators in slave devices as it uses a master’s clock.
* No complicated slave addressing system like I2C.
* Higher data transfer rate than I2C (almost twice as fast).
* Separate MISO and MOSI lines, so data can be sent and received at the same time.
* Simple software implementation.

**Disadvantages of SPI communication protocol**

* If there is more than one slave in communication then the wiring will be complex.
* Uses four wires (I2C and UARTs use two).
* No acknowledgment that the data has been successfully received (I2C has this).
* No form of error checking like the parity bit in UART.
* It only allows for a single master.

If you want to learn STM32 from scratch, you should follow this course “[**Mastering Microcontroller with Embedded Driver Development**](https://click.linksynergy.com/deeplink?id=UajS5soDmWY&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fmastering-microcontroller-with-peripheral-driver-development%2F)“. The course contains video lectures of 18.5-hours length covering all topics like, Microcontroller & Peripheral Driver Development for STM32 GPIO, I2C, SPI, USART using Embedded C.

**[](https://click.linksynergy.com/deeplink?id=UajS5soDmWY&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fmastering-microcontroller-with-peripheral-driver-development%2F)**

In the embedded system, I2C and SPI both play an important role. Both communication protocols are the example of synchronous communication but still, both have some important differences. In the below table, I have pointed out some common differences between SPI  and I2C ( SPI vs I2C ).

**The important difference between I2C and SPI ( I2C vs SPI ) communication protocol.**

|  |  |
| --- | --- |
| **I2C** | **SPI** |
| I2C can be multi-master and multi-slave, which means there can be more than one master and slave attached to the I2C bus. | SPI can be multi-save but does not a multi-master serial protocol, which means there can be only one master attached to the SPI bus. |
| I2C is a half-duplex communication protocol. | SPI is a full-duplex commination protocol. |
| I2C has the feature of clock stretching, which means if the slave cannot able to send fast data as fast enough then it suppresses the clock to stop the communication. | Clock stretching is not the feature of SPI. |
| I2C is used only two wire for the communication, one wire is used for the data and the second wire is used for the clock. | SPI needs three or four-wire for communication ((depends on requirement), MOSI, MISO, SCL, and Chip-select pin. |
| I2C is slower than SPI. | In comparison to I2C, SPI is faster. |
| I2C draws more power than SPI. | Draws less power as compared to I2C. |
| I2C is less susceptible to noise than SPI. | SPI is more susceptible to noise than I2C. |
| I2C is cheaper to implement than the SPI communication protocol. | Costly as compared to I2C. |
| I2C work on wire and logic and it has a pull-up resistor. | There is no requirement of a pull-up resistor in the case of the SPI. |
| In I2C communication we get the acknowledgment bit after each byte. | Acknowledgment bit is not supported by the SPI communication protocol. |
| I2C ensures that the data sent is received by the slave device. | SPI does not verify that data is received correctly or not. |
| I2C support multi-master communication. | SPI does not support multi-master communication. |
| I2C is a multi-master communication protocol that’s why it has the feature of arbitration. | SPI is not a multi-master communication protocol, so it does not consist of the properties of arbitration. |
| I2C is the address base bus protocol, you have to send the address of the slave for the communication. | In the case of the SPI, you have to select the slave using the slave select pin for the communication. |
| I2C has some extra overhead due to start and stop bits. | SPI does not have a start and stop bits. |
| I2C supports multiple devices on the same bus without any additional select lines (work on the basis of device address). | SPI requires additional signal (slave select lines) lines to manage multiple devices on the same bus. |
| I2C is better for long-distance. | SPI is better for a short distance. |
| I2C is developed by NXP. | SPI is developed by Motorola. |

The number of SPI slaves is not limited. In fact, you can add digital muxes (multiplexors) and control more slave devices than you have digital pins on the Arduino.

However, the SPI was design for communication over short distances within a box. So, the physical size of the bus can become a problem (bus capacitance, EMI). If you have to make a long-distance ruggedized SPI, there are application notes on the subject: [Extending the SPI bus for long-distance communication](http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=slyt441).

# Isolated SPI Communication Made Easy

[by **Thomas Brand**](https://www.analog.com/en/technical-articles/isolated-spi-communication-made-easy.html#author)[**Download PDF**](https://www.analog.com/media/en/technical-documentation/tech-articles/Isolated-SPI-Communication-Made-Easy.pdf)

Monitoring and controlling diverse systems requires direct access to the sensors and actuators, preferably from a central location and by means of a standardized communication method such as the serial peripheral interface (SPI). SPI is a synchronous serial data bus that facilitates data exchange between the devices and the central control unit over long distances. The communication takes place according to the master-slave principle and is full duplex. The SPI interface is made up of three lines: SDI, SDO, and SCK.

While the SPI communication method is generally suitable for distances up to approximately 10 m, to bridge longer distances, a repeater is often needed because of attenuation due to the increased line resistance of long cables. These signals must be amplified again. This also allows a larger signal-to-noise ratio (SNR) to be achieved at the same time. A device such as the isoSPI communication interface IC [LTC6820](https://www.analog.com/en/products/ltc6820.html) from Analog Devices, Inc. (ADI) can be used to read the signals.

Thanks to its innovative design, SPI communication can be maximized relatively easily by adding galvanic isolation using a twisted pair cable and appropriate transformers.

Because of harsh conditions frequently found in industrial environments, galvanically isolated communication components are often required to protect users from dangerous voltages as well as to ensure system reliability. Furthermore, isolation enables exact measurements despite occasional common-mode voltages. The isolation barrier is thereby key in separating the input stage from the rest of the system while still enabling a connection.

Figure 1 shows how all slaves are controlled by a single master. Masters and slaves can be microcontrollers or ADCs with an SPI interface to which the sensors, or microcontrollers, are usually connected. The LTC6820 thus enables bidirectional data transmission required for SPI communication between two completely galvanically isolated devices. It encodes the SPI signals from the master into differential signals with up to 1 Mbps, which are then transmitted through the galvanic isolation barrier and the twisted pair cables. At the opposite end of the cable, the differential signals are received again by an LTC6820 and decoded into SPI signals, which are then routed to the slave bus. The LTC6820 additionally supplies the required currents needed to drive the signals across the isolation barrier. Through an external resistor, these currents can be adapted to system requirements such as desired cable length, SNR, and immunity.

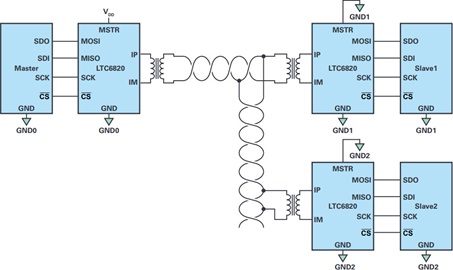
[[](https://www.analog.com/-/media/analog/en/landing-pages/technical-articles/isolated-spi-communication-made-easy/figure1.jpg?w=900&la=en&vs=1)](https://www.analog.com/-/media/analog/en/landing-pages/technical-articles/isolated-spi-communication-made-easy/figure1.jpg?w=900&la=en&vs=1)

Figure 1. Isolated SPI interface for control of multiple boards (slaves) via a common master.

However, note that despite the use of SPI repeaters, the data rate is limited and dependent on cable length. For example, the data rate for the circuit shown in Figure 1 with a 100 m CAT5 cable is only about 0.5 Mbps, which is half of the maximum possible value of 1 Mbps the LTC6820 could provide (see Figure 2).

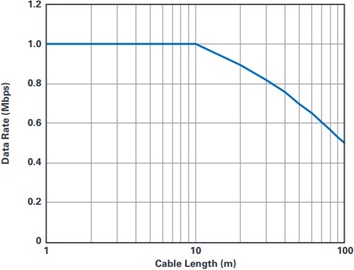
[[](https://www.analog.com/-/media/analog/en/landing-pages/technical-articles/isolated-spi-communication-made-easy/figure2.jpg?w=900&la=en&vs=1)](https://www.analog.com/-/media/analog/en/landing-pages/technical-articles/isolated-spi-communication-made-easy/figure2.jpg?w=900&la=en&vs=1)

Figure 2. Data rate in relation to cable length with use of a CAT5 cable.

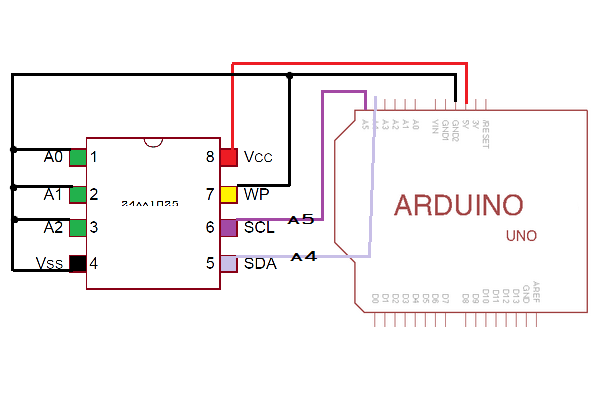
By using *iso*SPI communication ICs, the complexity of circuits for isolated transmission of SPI communication signals over long distances can be simplified, as a significant number of components typically required in conventional circuits can be omitted. In addition, with the LTC6820, distances of up to 100 m, which are not rare in industrial settings, can be realized. The LTC6820 makes it easy to implement daisy-chained applications in which one master controls several slaves. Furthermore, it is an ideal device for battery monitoring systems as they require galvanically isolated communication due to their partially explosive charge units (for example, lithium-ion batteries).

# I2C SERIAL CONNECTION: EEPROM AND ARDUINO:

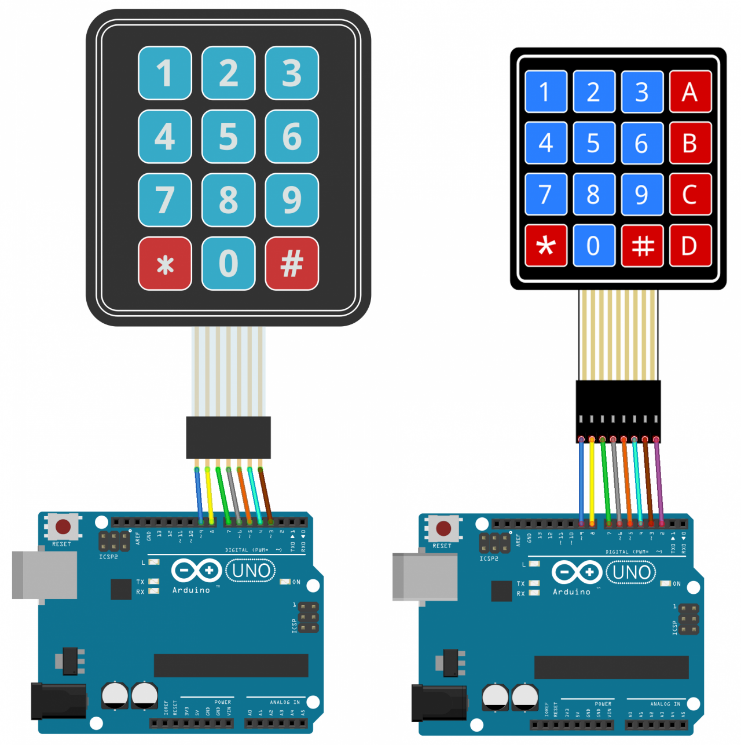
* **A5 – SCL**
* **A4 – SDA**

**FOR ARDUINO MEGA:**

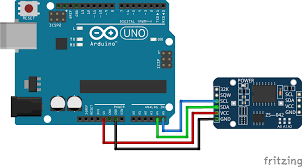
* **20 – SDA**
* **21 - SCL**



# Keypad connection to Arduino:



# Real time Clock Connection to Arduino:



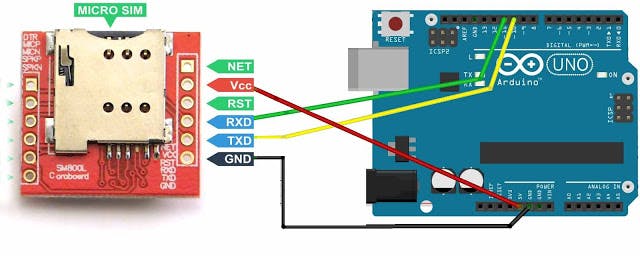
Both **i2c**-interface real time clock module and 2x16 LCD module use the same pin A4 (SDA) and A5 (SCL) on **Arduino** Uno. After hours of searching on the net the **i2c** bus **can** actually take many serial devices. This **is** possible because each device has its own unique address.

# Connecting Sim800L to Arduino:

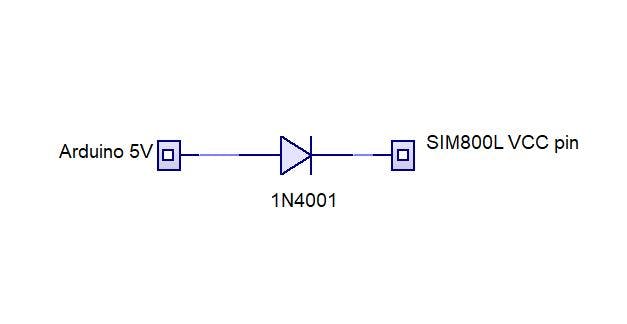
Let's see the wiring as the wiring part is the most important part. It may be a cause to burn the device since a single mistake.

[](javascript:openLightBox('804f7e6682',%200);)

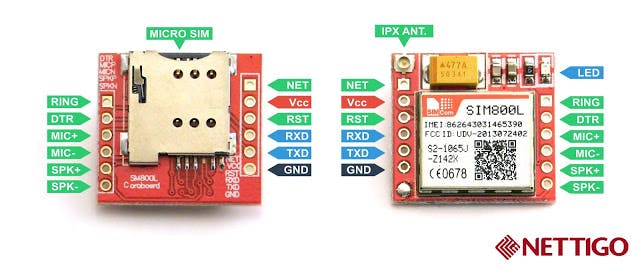
Connect the Arduino with the SIM800L modul, according to the above and connection will be shown as follows.

[](javascript:openLightBox('63846ce17b',%200);)

For safety, give the power to SIM800L module through Arduino Uno board via a diode as follows.

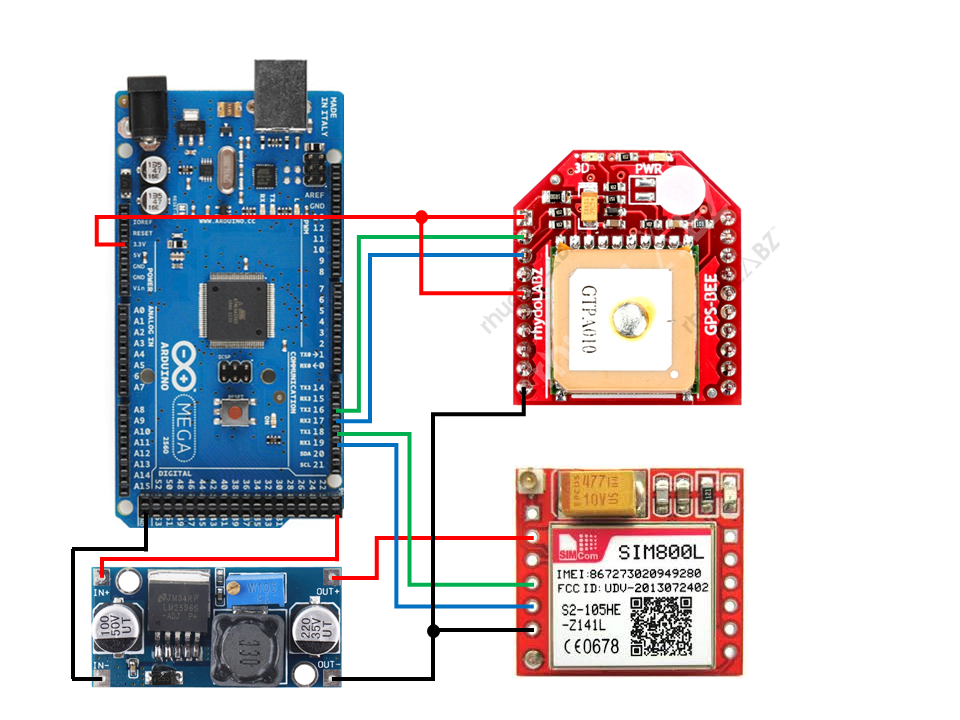
[](javascript:openLightBox('e5995fcf78',%200);)

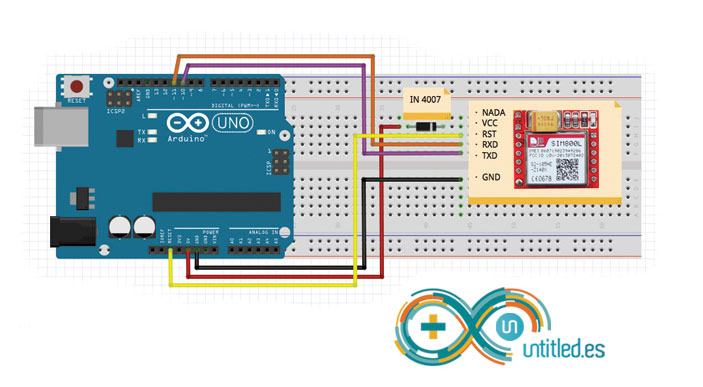
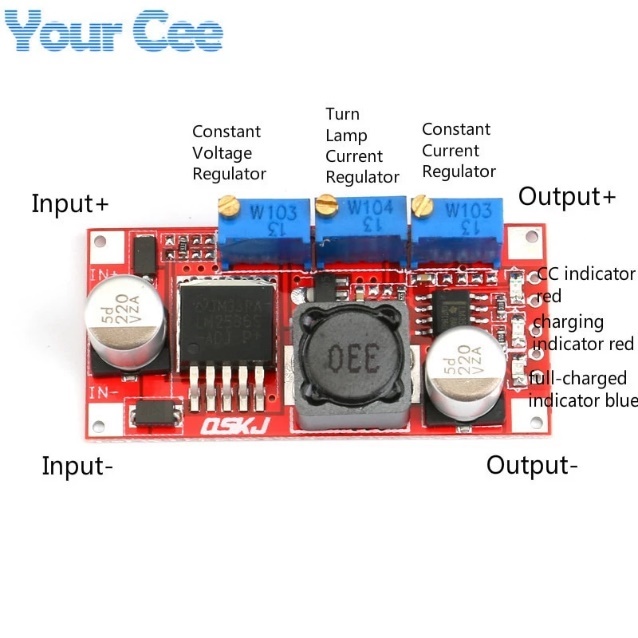
Fore more details read the following image as well.

[](javascript:openLightBox('87e18f678d',%200);)

If all the wiring connections are checked and confirmed as correct, go to the Arduino program and do the coding. Before coding, you must identify the correct direction of the SIM card. It is normally printed on the surface of the SIM slot.

Since **SIM800L** module doesn't come with onboard **voltage** regulator, an external power **supply** adjusted to **voltage** between 3.4V to 4.4V (Ideal 4.1V) is required.





# A Brief Introduction to the Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically there are three lines common to all the devices:

* **MISO** (Master In Slave Out) - The Slave line for sending data to the master,
* **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals,
* **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master

and one line specific for every device:

* **SS** (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.

When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

To write code for a new SPI device you need to note a few things:

* What is the maximum SPI speed your device can use? This is controlled by the first parameter in SPISettings. If you are using a chip rated at 15 MHz, use 15000000. Arduino will automatically use the best speed that is equal to or less than the number you use with SPISettings.
* Is data shifted in Most Significant Bit (MSB) or Least Significant Bit (LSB) first? This is controlled by second SPISettings parameter, either MSBFIRST or LSBFIRST. Most SPI chips use MSB first data order.
* Is the data clock idle when high or low? Are samples on the rising or falling edge of clock pulses? These modes are controlled by the third parameter in SPISettings.

The SPI standard is loose and each device implements it a little differently. This means you have to pay special attention to the device's datasheet when writing your code.

Generally speaking, there are four modes of transmission. These modes control whether data is shifted in and out on the rising or falling edge of the data clock signal (called the clock **phase**), and whether the clock is idle when high or low (called the clock **polarity**). The four modes combine polarity and phase according to this table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode** | **Clock Polarity (CPOL)** | **Clock Phase (CPHA)** | **Output Edge** | **Data Capture** |
| SPI\_MODE0 | 0 | 0 | Falling | Rising |
| SPI\_MODE1 | 0 | 1 | Rising | Falling |
| SPI\_MODE2 | 1 | 0 | Rising | Falling |
| SPI\_MODE3 | 1 | 1 | Falling | Rising |

Once you have your SPI parameters, use SPI.beginTransaction() to begin using the SPI port. The SPI port will be configured with your all of your settings. The simplest and most efficient way to use SPISettings is directly inside SPI.beginTransaction(). For example:

SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI\_MODE0));

If other libraries use SPI from interrupts, they will be prevented from accessing SPI until you call SPI.endTransaction(). The SPI settings are applied at the **begin** of the transaction and SPI.endTransaction() **doesn't change** SPI settings. **Unless** you, or some library, **calls** beginTransaction a second time, the setting are **maintained**. You should attempt to minimize the time between before you call SPI.endTransaction(), for best compatibility if your program is used together with other libraries which use SPI.

With most SPI devices, after SPI.beginTransaction(), you will write the slave select pin LOW, call SPI.transfer() any number of times to transfer data, then write the SS pin HIGH, and finally call SPI.endTransaction().

For more on SPI, see [Wikipedia's page on SPI](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Mode_Numbers).

### Connections

The following table display on which pins the SPI lines are broken out on the different Arduino boards:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Arduino / Genuino Board** | **MOSI** | **MISO** | **SCK** | **SS (slave)** | **SS (master)** | **Level** |
| Uno or Duemilanove | 11 or ICSP-4 | 12 or ICSP-1 | 13 or ICSP-3 | 10 | - | 5V |
| Mega1280 or Mega2560 | 51 or ICSP-4 | 50 or ICSP-1 | 52 or ICSP-3 | 53 | - | 5V |
| Leonardo | ICSP-4 | ICSP-1 | ICSP-3 | - | - | 5V |
| Due | SPI-4 | SPI-1 | SPI-3 | - | 4, 10, 52 | 3,3V |
| Zero | ICSP-4 | ICSP-1 | ICSP-3 | - | - | 3,3V |
| 101 | 11 or ICSP-4 | 12 or ICSP-1 | 13 or ICSP-3 | 10 | 10 | 3,3V |
| MKR1000 | 8 | 10 | 9 | - | - | 3,3V |

Note that MISO, MOSI, and SCK are available in a consistent physical location on the ICSP header; this is useful, for example, in designing a shield that works on every board.



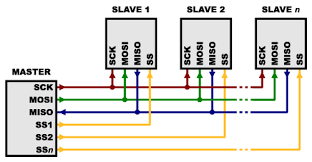
#### Note about Slave Select (SS) pin on AVR based boards

All AVR based boards have an SS pin that is useful when they act as a **slave** controlled by an external master. Since this library supports only master mode, this pin should be set always as OUTPUT otherwise the SPI interface could be put automatically into slave mode by hardware, rendering the library inoperative.

It is, however, possible to use any pin as the Slave Select (SS) for the devices. For example, the Arduino Ethernet shield uses pin 4 to control the SPI connection to the on-board SD card, and pin 10 to control the connection to the Ethernet controller.

### Examples

* [Barometric Pressure Sensor](https://www.arduino.cc/en/Tutorial/BarometricPressureSensor): Read air pressure and temperature from a sensor using the SPI protocol.
* [Digital Pot Control](https://www.arduino.cc/en/Tutorial/DigitalPotControl): Control a AD5206 digital potentiometer using the SPI protocol.



# [****PZEM-004T V3****](http://s.click.aliexpress.com/e/byNeshIC)



In many electrical projects, engineer directly deals with measurements with few basic requirements like.

* High galvanic isolation
* Parameter display
* Direct communication with computer
* Data acquisition and storage with subsequent viewing or copying to the computer.

**PZEM-004T** V3.0 or Version 3.0 is the upgraded version to replace the old **PZEM004T** V1.0. The old version has been sold out in most of the online store and no produce anymore. The updated version of PZEM004T is best for the DIY project, where we need to measure the voltage, current, power, energy, frequency, Power factor (frequency and PF is extra added in the new version) using Arduino/ESP8266/Raspberry Pi like open-source platform.



This module comes with 3 different current measurement options

* 10A range with a built-in Shunt resistor
* 100A external[closed Current Transformer](http://s.click.aliexpress.com/e/_sZAdwi)
* 100A[external split current Transformer](http://s.click.aliexpress.com/e/_sk8Jka)

I recommend the[**shell protected**](http://s.click.aliexpress.com/e/_s7xZGa)**split current transformer** module for a project like **Portable AC energy meter,** Energy meter for electrical Lab, Energy auditing and measuring equipment etc. as split current Transformer is more flexible than closed one. Closed CT is a low-cost option suitable for fixed type projects like Residential Energy management system, Load control system,**IoT based smart Energy meter, etc.**

The physical dimensions of the **PZEM-004T v3** board is 3.01×7.3 cm, The pzem-004t  V3 module is bundled with 33mm diameter 100A current transformer coil.

The new version has higher precision, faster refresh speed, and more stability communication than the old version.

## Voltage

* Measuring range:80～260V
* Resolution: 0.1V
* Measurement accuracy: 0.5%

## Power factor

* Measuring range: 0.00～1.00
* Resolution: 0.01
* Measurement accuracy: 1%

## Frequency

* Measuring range: 45Hz～65Hz
* Resolution: 0.1Hz
* Measurement accuracy: 0.5%

## Current

* Measuring range: 0～10A(PZEM-004T-10A); 0～100A(PZEM-004T-100A)
* Starting measure current: 0.01A(PZEM-004T-10A); 0.02A(PZEM-004T-100A)
* Resolution: 0.001A
* Measurement accuracy: 0.5%

## Active power

* Measuring range: 0～2.3kW(PZEM-004T-10A); 0～23kW(PZEM-004T-100A)
* Starting measure power: 0.4W
* Resolution: 0.1W
* Display format:

＜1000W, it displays one decimal, such as: 999.9W

≥1000W, it display only integer, such as: 1000W

* Measurement accuracy: 0.5%

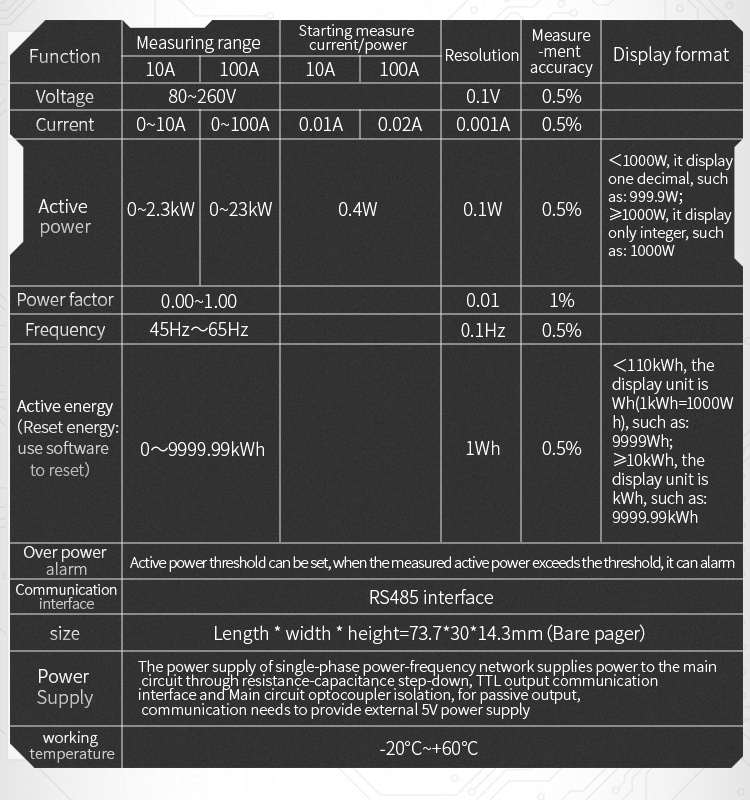
## Active Energy

* Measuring range: 0～9999.99kWh
* Resolution: 1Wh
* Measurement accuracy: 0.5%
* Display format:

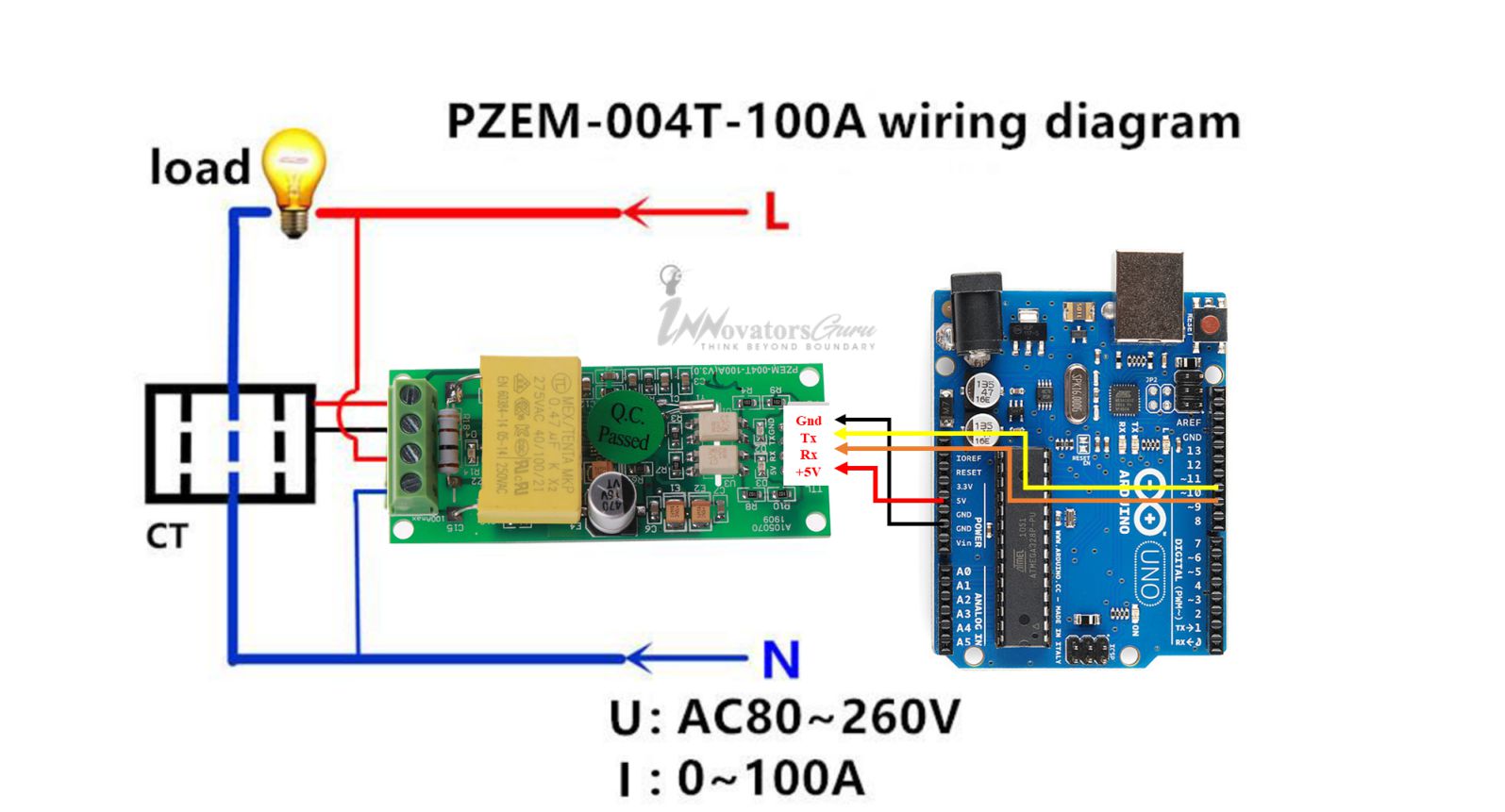
＜10kWh, the display unit is Wh(1kWh=1000Wh), such as: 9999Wh

≥10kWh, the display unit is kWh, such as: 9999.99kWh

## Function Description in short



## **PZEM-004T Arduino Wiring Diagram**

[](https://innovatorsguru.com/wp-content/uploads/2019/06/PZEM-004T-Arduino-2.jpg)

PZEM-004T V3 module has passive serial communication to connect Arduino Uno. Its RX and TX pin needs an external power supply. Connect the 5V pin to Arduino 5V pin, Rx pin to D11 of Arduino and Tx pin to D12 pin of Arduino, GND to GND. In the below diagram, PZEM004T V3 is connected to Arduino using Software serial. Use the following Program to test the module. Before compile download and install the [PZEM-004T V3 library.](https://github.com/mandulaj/PZEM-004T-v30)

# How nRF24L01+ Wireless Module Works & Interface with Arduino



Having two or more Arduino boards be able to communicate with each other wirelessly over a distance opens lots of possibilities like remotely monitoring sensor data, controlling robots, home automation and the list goes on. And when it comes down to having inexpensive yet reliable 2-way RF solutions, no one does a better job than nRF24L01+ transceiver module from [Nordic Semiconductor](http://www.nordicsemi.com/).

nRF24L01+ (plus) transceiver module can often be obtained online for less than two dollars, making it one of the most inexpensive data communication options that you can get. And best of all, these modules are super tiny, allowing you to incorporate a wireless interface into almost any project.

## **Hardware Overview**

### Radio Frequency

The nRF24L01+ transceiver module is designed to operate in 2.4 GHz worldwide ISM frequency band and uses [GFSK modulation](https://en.wikipedia.org/wiki/Frequency-shift_keying#Gaussian_frequency-shift_keying) for data transmission. The data transfer rate can be one of 250kbps, 1Mbps and 2Mbps.

### What is 2.4 GHz ISM band?

2.4 GHz band is one of the[Industrial, Scientific, and Medical (ISM) bands](https://en.wikipedia.org/wiki/ISM_band) reserved internationally for the use of unlicensed low-powered devices. Examples are Cordless phones, Bluetooth devices, near field communication (NFC) devices, and wireless computer networks (WiFi) all use the ISM frequencies.

### Power consumption

The operating voltage of the module is from **1.9 to 3.6V**, but the good news is that the **logic pins are 5-volt tolerant**, so we can easily connect it to an Arduino or any 5V logic microcontroller without using any logic level converter.

The module supports programmable output power viz. 0 dBm, -6 dBm, -12 dBm or -18 dBm and consumes unbelievably around **12 mA during transmission** at 0 dBm, which is even lower than a single LED. And best of all, it consumes 26 µA in standby mode and 900 nA at power down mode. That’s why they’re the go-to wireless device for low-power applications.

### SPI Interface

The nRF24L01+ transceiver module communicates over a 4-pin Serial Peripheral Interface (**SPI**) with a maximum data rate of **10Mbps**. All the parameters such as frequency channel (125 selectable channels), output power (0 dBm, -6 dBm, -12 dBm or -18 dBm), and data rate (250kbps, 1Mbps, or 2Mbps) can be configured through SPI interface.

The SPI bus uses a concept of a Master and Slave, in most common applications our Arduino is the Master and the nRF24L01+ transceiver module is the Slave. Unlike the I2C bus the number of slaves on the SPI bus is limited, on the Arduino Uno you can use a **maximum of two SPI slaves** i.e. two nRF24L01+ transceiver modules.

Here are complete specifications:

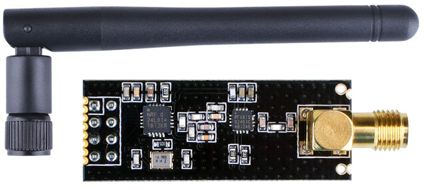
|  |  |
| --- | --- |
| Frequency Range | 2.4 GHz ISM Band |
| Maximum Air Data Rate | 2 Mb/s |
| Modulation Format | GFSK |
| Max. Output Power | 0 dBm |
| Operating Supply Voltage | 1.9 V to 3.6 V |
| Max. Operating Current | 13.5mA |
| Min. Current(Standby Mode) | 26µA |
| Logic Inputs | 5V Tolerant |
| Communication Range | 800+ m (line of sight) |

## **nRF24L01+ module Vs nRF24L01+ PA/LNA module**

There are a variety of modules available based upon the nRF24L01+ chip. Below are the most popular versions.

nRF24L01+ Wireless Module

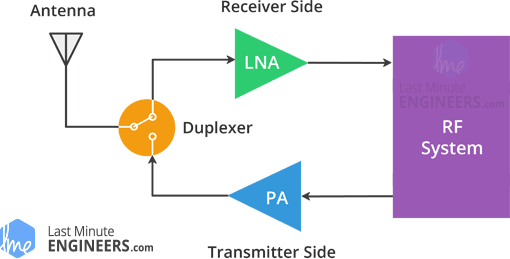
The first version uses on-board antenna. This allows for a more compact version of the breakout. However, the smaller antenna also means a lower transmission range. With this version, you’ll be able to communicate over a distance of **100 meters**. Of course that is outdoors in an open space. Your range indoors, especially through walls, will be slightly weakened.

nRF24L01+ PA LNA Wireless Transceiver Module with External Antenna

The second version comes with a SMA connector and a duck-antenna but that’s not the real difference. The real difference is that it comes with a special [RFX2401C chip](http://www.skyworksinc.com/Product/3213/RFX2401C) 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?

The PA stands for **Power Amplifier**. It merely boosts the power of the signal being transmitted from the nRF24L01+ chip. Whereas, LNA stands for **Low-Noise Amplifier**. The function of the LNA is to take the

nRF24L01+ PA/LNA Block Diagram

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)

The low-noise amplifier (LNA) of the receive path and the power amplifier (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. For more information check out this [article on digikey.com](https://www.digikey.com/en/articles/techzone/2013/oct/understanding-the-basics-of-low-noise-and-power-amplifiers-in-wireless-designs)

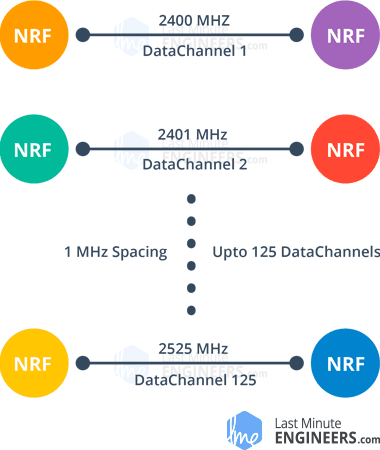
Except this difference, both modules are drop-in compatible. Meaning, if you build your project with one you can just unplug it and use another without need to make any changes to the system.

## **How nRF24L01+ transceiver module works?**

### RF Channel Frequency

The 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 or to be more precise, it could be between 2.400 to 2.525 GHz (2400 to 2525 MHz).

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.



The channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps air data rate. However at 2Mbps air data rate, 2MHz bandwidth is occupied (wider than the resolution of RF channel frequency setting). So, to ensure non-overlapping channels and reduce cross-talk in 2Mbps mode, you need to keep 2MHz spacing between two channels.

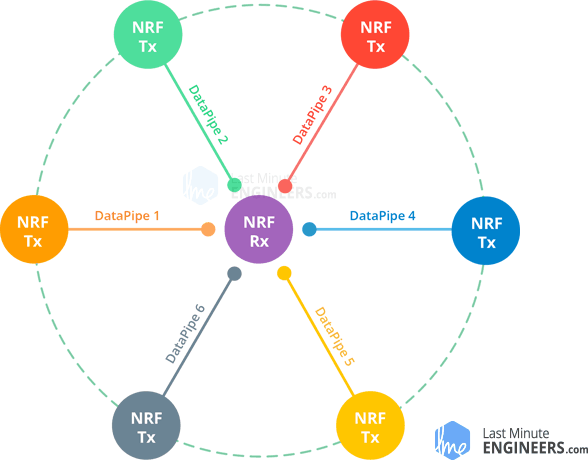
RF channel frequency of your selected channel is set according to the following formula:

Freq(Selected) = 2400 + CH(Selected)

For example, if you select 108 as your channel for data transmission, the RF channel frequency of your channel would be 2508MHz (2400 + 108)

### nRF24L01+ Multiceiver Network

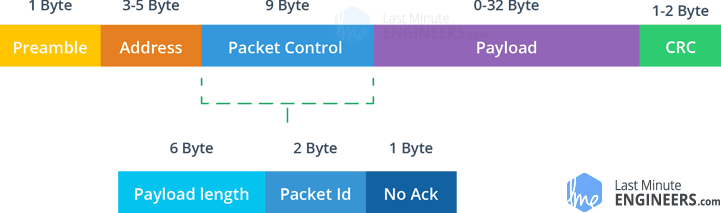
The nRF24L01+ provides a feature called **Multiceiver**. It’s an abbreviation for **Multi**ple Transmitters Single Re**ceiver**. In which each RF channel is logically divided into 6 parallel data channels called **Data Pipes**. In other words, a data pipe is a logical channel in the physical RF Channel. Each data pipe has its own physical address (Data Pipe Address) and can be configured. This can be illustrated as shown below.

nRF24L01+ Multiceiver Network – Multiple Transmitters Single Receiver

To simplify the above diagram, imagine the primary receiver acting as a hub receiver collecting information from 6 different transmitter nodes simultaneously. The hub receiver can stop listening any time and acts as a transmitter. But this can only be done one pipe/node at a time.

### Enhanced ShockBurst Protocol

The nRF24L01+ transceiver module uses a packet structure known as Enhanced ShockBurst. This simple packet structure is broken down into 5 different fields, which is illustrated below.

nRF24L01+ Enhanced ShockBurst Packet Structure

The original ShockBurst structure consisted only of Preamble, Address, Payload and the Cyclic Redundancy Check (CRC) fields. Enhanced ShockBurst brought about greater functionality for more enhanced communications using a newly introduced **Packet Control Field** (PCF).

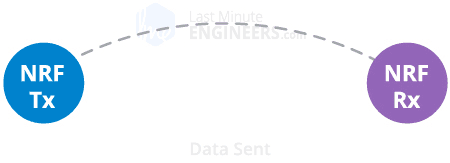
This new structure is great for a number of reasons. Firstly, it allows for variable length payloads with a payload length specifier, meaning payloads can vary from 1 to 32 bytes.

Secondly, it provides each sent packet with a packet ID, which allows the receiving device to determine whether a message is new or whether it has been retransmitted (and thus can be ignored).

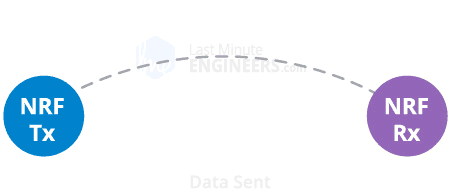
Finally, and most importantly, each message can request an acknowledgement to be sent when it is received by another device.

### nRF24L01+ Automatic Packet Handling

Now, let’s discuss three scenarios to get a better understanding of how two nRF24L01+ modules transact with each other.



**Transaction with acknowledgement and interrupt**This is an example of positive scenario. Here the transmitter starts a communication by sending a data packet to the receiver. Once the whole packet is transmitted, it waits (around 130 µs) for the acknowledgement packet (ACK packet) to receive. When the receiver receives the packet, it sends ACK packet to the transmitter. On receiving the ACK packet the transmitter asserts interrupt (IRQ) signal to indicate the new data is available.



**Transaction with data packet lost**This is a negative scenario where a retransmission is needed due to loss of the packet transmitted. After the packet is transmitted, the transmitter waits for the ACK packet to receive. If the transmitter doesn’t get it within Auto-Retransmit-Delay (ARD) time, the packet is retransmitted. When the retransmitted packet is received by the receiver, the ACK packet is transmitted which in turn generates interrupt at the transmitter.

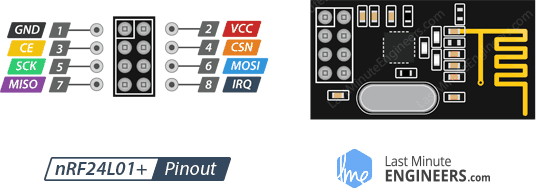


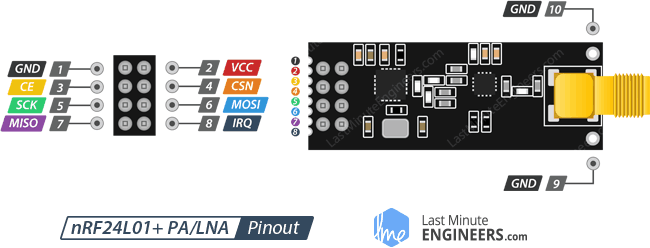
**Transaction with acknowledgement lost**This is again a negative scenario where a retransmission is needed due to loss of the ACK packet. Here even if the receiver receives the packet in the first attempt, due to the loss of ACK packet, transmitter thinks the receiver has not got the packet at all. So, after the Auto-Retransmit-Delay time is over, it retransmits the packet. Now when receiver receives the packet containing same packet ID as previous, it discards it and sends ACK packet again.

This whole packet handling is done automatically by the nRF24L01+ chip without involvement of the microcontroller.

## **nRF24L01+ Transceiver Module Pinout**

Let’s have a look at the pinout of both the versions of nRF24L01+ transceiver Module.





GND is the Ground Pin. It is usually marked by encasing the pin in a square so it can be used as a reference for identifying the other pins.

VCC supplies power for the module. This can be anywhere from 1.9 to 3.9 volts. You can connect it to 3.3V output from your Arduino. Remember connecting it to 5V pin will likely destroy your nRF24L01+ module!

CE (Chip Enable) is an active-HIGH pin. When selected the nRF24L01 will either transmit or receive, depending upon which mode it is currently in.

CSN (Chip Select Not) is an active-LOW pin and is normally kept HIGH. When this pin goes low, the nRF24L01 begins listening on its SPI port for data and processes it accordingly.

SCK (Serial Clock) accepts clock pulses provided by the SPI bus Master.

MOSI (Master Out Slave In) is SPI input to the nRF24L01.

MISO (Master In Slave Out) is SPI output from the nRF24L01.

IRQ is an interrupt pin that can alert the master when new data is available to process.

## **Wiring – Connecting nRF24L01+ transceiver module to Arduino UNO**

Now that we have a complete understanding of how nRF24L01+ transceiver module works, we can begin hooking it up to our Arduino!

To start with, connect VCC pin on the module to 3.3V on the Arduino and GND pin to ground. The pins CSN and CE can be connected to any digital pin on the Arduino. In our case, it’s connected to digital pin#8 and #9 respectively. Now we are remaining with the pins that are used for SPI communication.

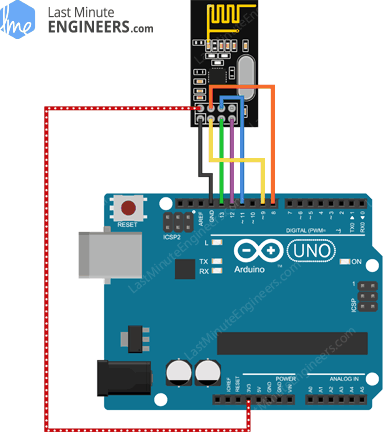
As nRF24L01+ transceiver module require a lot of data transfer, they will give the best performance when connected up to the hardware SPI pins on a microcontroller. The hardware SPI pins are much faster than ‘bit-banging’ the interface code using another set of pins.

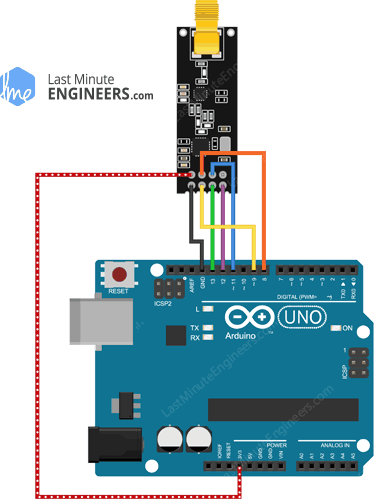
Note that each Arduino Board has different SPI pins which should be connected accordingly. For Arduino boards such as the UNO/Nano V3.0 those pins are digital 13 (SCK), 12 (MISO) and 11 (MOSI).

If you have a Mega, the pins are different! You’ll want to use digital 50 (MISO), 51 (MOSI), 52 (SCK), and 53 (SS). Refer below table for quick understanding.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MOSI | MISO | SCK |
| Arduino Uno | 11 | 12 | 13 |
| Arduino Nano | 11 | 12 | 13 |
| Arduino Mega | 51 | 50 | 52 |

In case you’re using different Arduino board than mentioned above, it is advisable to check the Arduino official [documentation](https://www.arduino.cc/en/Reference/SPI) before proceeding.

Wiring nRF24L01+ Wireless Transceiver Module to Arduino UNO

Wiring nRF24L01+ PA LNA Wireless Module to Arduino UNO

Remember! You need to make two of these circuits. One acts as a transmitter and the other as a receiver. The wiring for both is identical.

Once you have everything hooked up you are ready to go!

## **RF24 Arduino Library for nRF24L01+ Module**

Interfacing with nRF24L01+ transceiver module is a bunch of work, but luckily for us, there are a number of libraries available. One of the popular libraries is [RF24](http://tmrh20.github.io/RF24/). This library has been around for several years. It is simple to use for beginners, but yet offers a lot for advanced users. In our experiments, we will be using the same library.

You can download the latest version of library on the [RF24 GitHub repository fork](https://github.com/nRF24/RF24) or, just click this button to download the zip:

[RF24-master.zip](https://lastminuteengineers.com/libraries/RF24-master.zip)

To install it, open the Arduino IDE, go to Sketch > Include Library > Add .ZIP Library, and then select the RF24-master file that you just downloaded. If you need more details on installing a library, visit this [Installing an Arduino Library](https://www.arduino.cc/en/Guide/Libraries)tutorial.

## **Arduino Code – For Transmitter**

In our experiment we will just send a traditional ‘**Hello World**’ message from the transmitter to the receiver.

Here is the sketch we will be using for our transmitter:

//Include Libraries

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

//create an RF24 object

RF24 radio(9, 8); // CE, CSN

//address through which two modules communicate.

const byte address[6] = "00001";

void setup()

{

radio.begin();

//set the address

radio.openWritingPipe(address);

//Set module as transmitter

radio.stopListening();

}

void loop()

{

//Send message to receiver

const char text[] = "Hello World";

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

delay(1000);

}

The sketch starts by including the libraries. SPI.h library handles the SPI communication while nRF24L01.h and RF24.h controls the module.

//Include Libraries

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

Next, we need to create an RF24 object. The object takes two pin numbers as parameters to which signals CE and CSN are connected.

//create an RF24 object

RF24 radio(9, 8); // CE, CSN

Next we need to create a byte array which will represent the pipe address through which two nRF24L01+ modules communicate.

//address through which two modules communicate.

const byte address[6] = "00001";

We can change the value of this address to any 5-letter string such as “node1”. The address is necessary if you have a few modules in a network. Thanks to the address, you can choose a particular module to which you are interested in communicating, so in our case we will have the same address for both the transmitter and the receiver.

Next in the setup function: we need to initialize the radio object using radio.begin() and using the radio.openWritingPipe() function we set the address of the transmitter.

//set the address

radio.openWritingPipe(address);

Finally, we will use the radio.stopListening() function which sets module as transmitter.

//Set module as transmitter

radio.stopListening();

In the loop section: we create an array of characters to which we assign the message “Hello World”. Using the radio.write() function we will send that message to the receiver. The first argument here is the message that we want to send. The second argument is the number of bytes present in that message.

const char text[] = "Hello World";

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

Through this method, you can send up to 32 bytes at a time. Because that is the maximum size of a single packet nRF24L01+ can handle. If you need a 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, the data has been lost.

the radio.write() function blocks the program until it receives the acknowledgment or runs out of all attempts of retransmission.

## **Arduino Code – For Receiver**

Here is the sketch we will be using for our receiver

//Include Libraries

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

//create an RF24 object

RF24 radio(9, 8); // CE, CSN

//address through which two modules communicate.

const byte address[6] = "00001";

void setup()

{

while (!Serial);

Serial.begin(9600);

radio.begin();

//set the address

radio.openReadingPipe(0, address);

//Set module as receiver

radio.startListening();

}

void loop()

{

//Read the data if available in buffer

if (radio.available())

{

char text[32] = {0};

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

Serial.println(text);

}

}

This program looks quite similar to the program of the transmitter except some changes.

At the beginning of the setup function we start the serial communication. Next using radio.setReadingPipe() function we set the same address as transmitter and in that way we enable the communication between transmitter and receiver.

//set the address

radio.openReadingPipe(0, address);

The first argument is the number of the stream. You can create up to 6 streams that respond 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.

The next step is to set the module as a receiver and start receiving data. To do that we use radio.startListening() function. From that moment the modem waits for data sent to the specified address.

//Set module as receiver

radio.startListening();

In the loop function: The sketch checks whether any data has arrived at the address using radio.available() method. This method returns TRUE value if we any data is available in buffer.

if (radio.available())

{

char text[32] = {0};

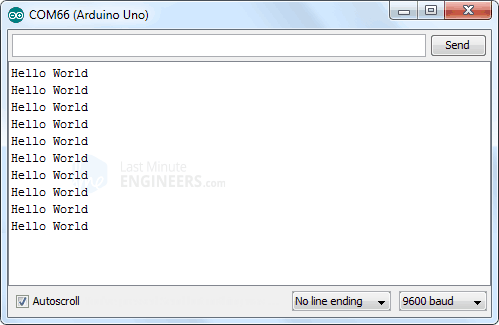
radio.read(&text, sizeof(text));

Serial.println(text);

}

If the data is received, then it creates an array of 32 characters 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)). This will store the received data in to our character array.

At the end we just print the received message on serial monitor. If you did everything ok and there are no mistakes in connections, you should see something like this in your Serial Monitor.

nRF24L01+ Transceiver Output on Serial Monitor

## **Improving range of nRF24L01+ transceiver Module**

A key parameter for a wireless communication system is the communication range. In many cases it’s the deciding factor for choosing an RF solution. So, let’s discuss what we can do to get a better range for our module.

### Reduce Power Supply Noise

An RF circuit that generates a Radio Frequency (RF) signal, is very sensitive to power supply noise. If not controlled, the power supply noise can significantly reduce the range you can get.

Unless the power source is a stand-alone battery, there is a good chance that there is noise associated with the generation of the power. To prevent this noise from entering the system, it is advised to place a **10 µf filter capacitor across the power supply** line as physically close to the nRF24L01+ module as possible.

An easiest way to get over with is to use a very inexpensive Adapter Module for nRF24L01.

nRF24L01+ Adapter

The adapter module has an 8-pin female connector to allow you to plug in your nRF24L01 module. It can accommodate both the module we discussed earlier, the one with integrated antenna and other with external antenna (PA/LNA). It also has a 6-pin male connector for the SPI and Interrupt connections and a 2-pin connector for power input.

The adapter module has its own 3.3 volt voltage regulator and a set of filter capacitors, so you can power it with a 5-volt power supply.

### Change your channel frequency

Another potential source of noise for an RF circuit is the outside environment, especially if you have neighboring networks set on the same channel or interference from other electronics.

To prevent these signals from causing issues, we suggest using the **highest 25 channels** your nRF24L01+ module. Reason for this is WiFi uses most of the lower channels.

### Lower Data Rate

The nRF24L01+ offers highest receiver sensitivity at 250Kbps speed which is -94dBm. However at 2MBps data rate, the receiver sensitivity drops to -82dBm. If you speak this language, you know that the receiver at 250Kbps is nearly 10 times more sensitive than at 2Mbps. That means the receiver can decode a signal that is 10 times weak.

### What does Receiver (Rx) sensitivity mean?

Receiver sensitivity is the lowest power level at which the receiver can detect an RF signal. The larger the absolute value of the negative number, the better the receiver sensitivity. For example, a receiver sensitivity of −94 dBm is better than a receiver sensitivity of −82 dBm by 12 dB.

So, lowering the data rate can significantly improve the range you can achieve. Also, for most of our projects, **250Kbps speed** is more than sufficient.

### Higher Output Power

Setting maximum output power can also improve the communication range. The nRF24L01+ lets you choose one of the output power viz. 0 dBm, -6 dBm, -12 dBm or -18 dBm. Selecting **0 dBm output power** sends stronger signal over the air.