**<https://circuitdigest.com/microcontroller-projects/raspberry-pi-with-lora-peer-to-peer-communication-with-arduino>**

**Este tutorial es usando librerias si utiliza el** RFM95 LoRa transceiver en vez del **433 LoRa SX1278**

**LoRa** is` getting increasingly popular with the advent of IoT, Connected Cars, M2M, Industry 4.0 etc. Because of its ability to communicate to long distances with very less power it is preferably used by designers to send/receive data from a battery powered Thing.

In this tutorial we will learn **how to use a LoRa module SX1278 with Raspberry pi** to communicate with another SX1278 connected to a microcontroller like Arduino.

Materials Required

* SX1278 433MHz LoRa Module – 2 Nos
* 433MHz LoRa antenna – 2Nos
* Arduino UNO- or other version
* Raspberry Pi 3

**Warning**:

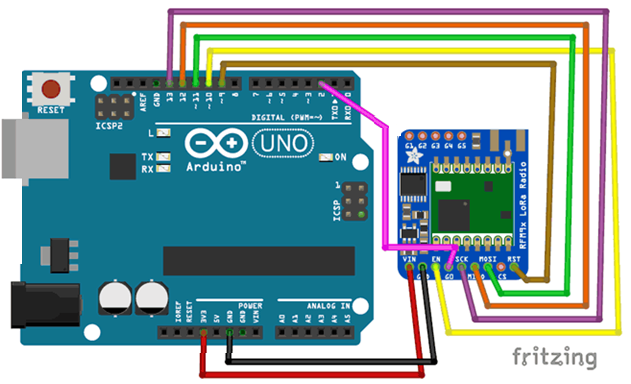
* Always use your SX1278 LoRa module with 433 MHz antennas; else the module might get damaged.
* Power the LoRa module with separate 3.3V power rail as the Pi might not be able to source enough current.

The **SX1278** is a 16-pin **Lora module** that communicates using SPI on 3.3V Logic.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Raspberry Pi** | **Lora – SX1278 Module** | | 3.3V | 3.3V | | Ground | Ground | | GPIO 10 | MOSI | | GPIO 9 | MISO | | GPIO 11 | SCK | | GPIO 8 | Nss / Enable | | GPIO 4 | DIO 0 | | GPIO 17 | DIO 1 | | GPIO 18 | DIO 2 | | GPIO 27 | DIO 3 | | GPIO 22 | RST | |  |

Connecting Arduino with LoRa

The connection for the Arduino module remains the same as that we used in our [previous tutorial](https://circuitdigest.com/microcontroller-projects/arduino-lora-sx1278-interfacing-tutorial). The only difference will be instead of using the library from [Sandeep Mistry](https://github.com/sandeepmistry/arduino-LoRa) we will use the Rspreal library based on Radio head which we will discuss later in this project. The circuit is give below



Again you can use the 3.3V pin on Arduino Uno or use a separate 3.3V regulator.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **LoRa SX1278 Module** | **Arduino UNO Board** | | 3.3V | 3.3V | | Gnd | Gnd | | En/Nss | D10 | | G0/DIO0 | D2 | | SCK | D13 | | MISO | D12 | | MOSI | D11 | | RST | D9 | | Arduino LoRa Module Hardware Setup |

pyLoRa for Raspberry Pi

There are many python packages that you can use with LoRa. Also commonly the Raspberry Pi is used as a LoRaWAN to get data from multiple LoRa nodes. But, in this project [pyLoRa package](https://pypi.org/project/pyLoRa/" \t "_blank).

It has a [rpsreal LoRa Arduino](https://github.com/rpsreal/LoRa_Ra-02_Arduino" \t "_blank) and [rpsreal LoRa Raspberry pi](https://github.com/rpsreal/pySX127x" \t "_blank) modules which can be used on the Arduino and the Raspberry Pi environment.

Configuring the Raspberry Pi for LoRa module

* Enable protocol SPI
* install the ***pylora*package**.

**Step 1:** Get into the **configuration window** using the following command. To get the below window

**sudo raspi-config**

**Step 2:**  Navigate to interfacing options and enable SPI.

Text

Description automatically generated with low confidence

**Step 3:**Save the changes and get back to the terminal window.

Make sure pip and python is updated and then **install the *RPi.GPIO* package** using the following command.

**pip install RPi.GPIO**

This package class will help us control the GPIO pin on the Pi.

**Step 4: Install** ***spidev* package** using the following command.

Spidev is a python binding for Linux which can be used to perform SPI communication on Raspberry Pi.

**pip install spidev**

**Step 5: Install the pyLoRa package**

**pip install pyLoRa**

The PyLoRa package also supports encrypted communication which can be used with Arduino and Raspberry Pi seamlessly.

This will improve the data security in your communication. But you have to install separate package after this step which I am not doing since encryption is not in the scope of this tutorial.

You can follow the above github links for more details.

After, this step you can add the package path information to pi and try with the **python program given at the end**. But I was not able to add the path successfully and hence had to manually download library and use the same directly for my programs. So I had to proceed with the following steps

**Step 6:**Download and **install the python-rpi.gpio package and spidev package** using the below command.

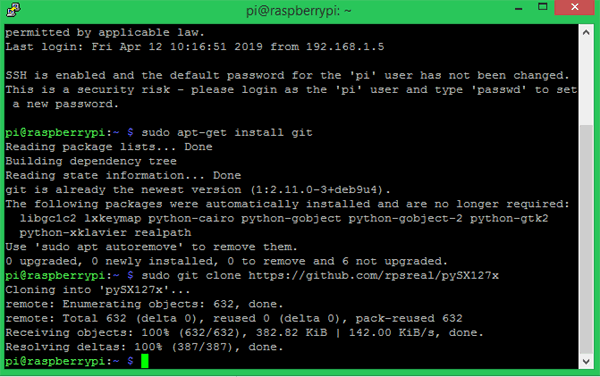
**sudo apt-get install python-rpi.gpio python3-rpi.gpio**

**sudo apt-get install python-spidev python3-spidev**

**Step 7:** Also install git and then use it to clone the python directory for our Raspberry Pi. You can do that using the following commands.

**sudo apt-get install git**

**sudo git clone https://github.com/rpsreal/pySX127x**



Once this step is complete **you should find the SX127x sub directory in Raspberry Pi home folder**. This will have all the required files associated with the library.

Programming Raspberry Pi for LoRa

In a peer to peer LoRa communication the module that is transmitting the information is called a server and the module that receives the information is called a client. In most cases the Arduino will be used in the field with a sensor to measure data and the Pi will be used to receive these data. So, I decided to use the **Raspberry Pi as a client and the Arduino as a server** in this tutorial. The **complete Raspberry Pi client program can be found at the bottom of this page**. Here I will try to explain the important lines in the program.

**Caution:** Make sure the program file is in the same directory where the SX127x library folder is present. You can copy this folder and use it anywhere if you wish to port the project.

S**et the LoRa module to work in 433Mhz and then listen for incoming packets.** If we receive anything we simple print them on the console.

**from time import sleep // to create delays**

**from SX127x.LoRa import \* // for LoRa Communication**

**from SX127x.board\_config import BOARD // Set the board and LoRa parameters**

**BOARD.setup()**

Next we **create the python LoRa class** with three functions

1. init class,
2. start class
3. o*n\_rx\_done* class

**Init class** initializes the LoRa module in 433MHz with 125KHz bandwidth, then put in sleep mode

**# Medium Range Defaults after init are 434.0MHz, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on 13 dBm**

**lora.set\_pa\_config(pa\_select=1)**

**def \_\_init\_\_(self, verbose=False):**

**super(LoRaRcvCont, self).\_\_init\_\_(verbose)**

**self.set\_mode(MODE.SLEEP)**

**self.set\_dio\_mapping([0] \* 6)**

T**he start function** to **configure the module as receiver and obtain like RSSI (Receiving signal strength Indicator), status, operating frequency etc**.

We set the module to work in continuous receiver mode (RXCONT) from sleep mode and then use a while loop to read values like RSSI and modem status. We also flush the data in the serial buffer onto the terminal.

**def start(self):**

**self.reset\_ptr\_rx()**

**self.set\_mode(MODE.RXCONT)**

**while True:**

**sleep(.5)**

**rssi\_value = self.get\_rssi\_value()**

**status = self.get\_modem\_status()**

**sys.stdout.flush()**

**Finally the *on\_rx\_done* function gets executed after the incoming packet is read**.

In this function the received values is moved into a variable called payload from the Rx buffer after setting the receiving flag high. Then the received values are decoded with utf-8 to print a user readable data on the shell. We also put the module back in sleep mode till another value is received.

**def on\_rx\_done(self):**

**self.clear\_irq\_flags(RxDone=1)**

**payload = self.read\_payload(nocheck=True)**

**print("\nReceived: ")**

**print(bytes(payload).decode("utf-8",'ignore'))**

**self.set\_mode(MODE.SLEEP)**

**self.reset\_ptr\_rx()**

**self.set\_mode(MODE.RXCONT)**

The remaining part of the program is just to **print the received values on the console and terminate the program using a keyboard interrupt**. We again set the board in sleep mode even after termination of the program to save power.

try:

lora.start()

except KeyboardInterrupt:

sys.stdout.flush()

print("")

sys.stderr.write("KeyboardInterrupt\n")

finally:

sys.stdout.flush()

**print("")**

lora.set\_mode(**MODE.SLEEP**)

BOARD.teardown()

Arduino Code for LoRa to communicate with Raspberry Pi

As I mentioned earlier the ***rpsreal*** code supports both Arduino and Pi and hence communication between Arduino and Pi is possible. It works based on the Radiohead Library from AirSpayce’s. So you have to **install the radio head library first to your Arduino IDE**.

To do that visit the [Github page](https://github.com/PaulStoffregen/RadioHead" \t "_blank) and download the library in ZIP folder. Then place it in the library folder of your Arduino IDE.  Now, restart the Arduino IDE and you will find example files for Radio head library. **Here we will program the Arduino to work as a LoRa server** to send test packets like 0 to 9.

We begin the program by importing the SPI library (installed by default) to use SPI protocol and then the **RH\_RF95** library from Radio head to perform LoRa communication. Then we define to which pin of Arduino we have connected the Chip select (CS), Reset (RST) and Interrupt (INT) pin of the LoRa with Arduino. Finally we also define that the module should work in 434MHz Frequency and initialize the LoRa module.

**#include <SPI.h> //Import SPI library**

**#include <RH\_RF95.h> // RF95 from RadioHead Library**

**#define RFM95\_CS 10 //CS if Lora connected to pin 10**

**#define RFM95\_RST 9 //RST of Lora connected to pin 9**

**#define RFM95\_INT 2 //INT of Lora connected to pin 2**

**// Change to 434.0 or other frequency, must match RX's freq!**

**#define RF95\_FREQ 434.0**

**// Singleton instance of the radio driver**

**RH\_RF95 rf95(RFM95\_CS, RFM95\_INT);**

Inside the*setup*function we will reset the **LoRa module by pulling its reset pin to low** for 10 milli second to start fresh. Then we **initialize it with the module** that we created earlier using Radio head library. Then, we **set the frequency and transmission power for the LoRa server**. Higher the transmission more distance your packets will travel but will consume more power.

**void setup()**

**{**

**Serial.begin(9600);**

**// Reset LoRa Module**

**pinMode(RFM95\_RST, OUTPUT);**

**digitalWrite(RFM95\_RST, LOW);**

**delay(10);**

**digitalWrite(RFM95\_RST, HIGH);**

**delay(10);**

**//Initialize LoRa Module**

**while (!rf95.init()) {**

**Serial.println("LoRa radio init failed");**

**while (1);**

**}**

**//Set the default frequency 434.0MHz**

**if (!rf95.setFrequency(RF95\_FREQ)) {**

**Serial.println("setFrequency failed");**

**while (1);**

**}**

**rf95.setTxPower(18); //Transmission power of the Lora Module**

**}**

**Inside the infinite *loop* function, we simply have to send the data packet through the LoRa module.**This data can be anything like sensor value of user command. But for simplicity we will send char value 0 to 9 for every 1 second interval and then initialize the value back to 0 after reaching 9. Note that the values can be sent only in a char array format and the type of data should be unit8\_t that is 1 byte at a time. The code to do the same is shown below

**void loop()**

**{**

**Serial.print("Send: ");**

**char radiopacket[1] = char(value)};**

**rf95.send((uint8\_t \*)radiopacket, 1);**

**delay(1000);**

**value++;**

**if (value > '9')**

**value = 48;**

**}**

Testing LoRa Communication between Raspberry Pi and Arduino

Once the python client sketch is launched on the Pi (use only python 3). You should notice “Received: 0” to 9 like shown in the image below.

Received: 7

Received: 8

Received: 9

Received: 0

Received: 1

The [complete Raspberry pi code with all the required libraries can be downloaded from here](https://circuitdigest.com/sites/default/files/Raspberry-pi-lora-code.zip).

Code

**Lora Receiver Code for Raspberry Pi:**([Download the required libraries from here](https://circuitdigest.com/sites/default/files/Raspberry-pi-lora-code.zip))

from time import sleep  
from SX127x.LoRa import \*  
from SX127x.board\_config import BOARD

BOARD.setup()

class LoRaRcvCont(LoRa):  
    def \_\_init\_\_(self, verbose=False):  
        super(LoRaRcvCont, self).\_\_init\_\_(verbose)  
        self.set\_mode(MODE.SLEEP)  
        self.set\_dio\_mapping([0] \* 6)

    def start(self):  
        self.reset\_ptr\_rx()  
        self.set\_mode(MODE.RXCONT)  
        while True:  
            sleep(.5)  
            rssi\_value = self.get\_rssi\_value()  
            status = self.get\_modem\_status()  
            sys.stdout.flush()

    def on\_rx\_done(self):  
        print("\nReceived: ")  
        self.clear\_irq\_flags(RxDone=1)  
        payload = self.read\_payload(nocheck=True)  
        print(bytes(payload).decode("utf-8",'ignore'))  
        self.set\_mode(MODE.SLEEP)  
        self.reset\_ptr\_rx()  
        self.set\_mode(MODE.RXCONT)

lora = LoRaRcvCont(verbose=False)  
lora.set\_mode(MODE.STDBY)

#  Medium Range  Defaults after init are 434.0MHz, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on 13 dBm

lora.set\_pa\_config(pa\_select=1)

try:  
    lora.start()  
except KeyboardInterrupt:  
    sys.stdout.flush()  
    print("")  
    sys.stderr.write("KeyboardInterrupt\n")  
finally:  
    sys.stdout.flush()  
    print("")  
    lora.set\_mode(MODE.SLEEP)  
    BOARD.teardown()

**Lora Server Code for Arduino:**

//Arduino Raspberry Pi wireless Comunnication through LoRa - SX1278  
//Send 0 to 9 from Arduino through Radio head LoRa without ACK

#include <SPI.h>   
#include <RH\_RF95.h> // RF95 from RadioHead Library

#define RFM95\_CS 10 //CS if Lora connected to pin 10  
#define RFM95\_RST 9 //RST of Lora connected to pin 9  
#define RFM95\_INT 2 //INT of Lora connected to pin 2

// Change to 434.0 or other frequency, must match RX's freq!  
#define RF95\_FREQ 434.0

// Singleton instance of the radio driver  
RH\_RF95 rf95(RFM95\_CS, RFM95\_INT);

void setup()   
{  
   Serial.begin(9600);  
    
// Reset LoRa Module   
  pinMode(RFM95\_RST, OUTPUT);   
  digitalWrite(RFM95\_RST, LOW);  
  delay(10);  
  digitalWrite(RFM95\_RST, HIGH);  
  delay(10);

//Initialize LoRa Module  
  while (!rf95.init()) {  
    Serial.println("LoRa radio init failed");  
    while (1);  
  }

 //Set the default frequency 434.0MHz  
  if (!rf95.setFrequency(RF95\_FREQ)) {  
    Serial.println("setFrequency failed");  
    while (1);  
  }

  rf95.setTxPower(18); //Transmission power of the Lora Module  
}

char value = 48;

void loop()  
{  
  Serial.print("Send: ");  
  char radiopacket[1] = char(value)};  
  rf95.send((uint8\_t \*)radiopacket, 1);

  delay(1000);  
  value++;  
  if (value > '9')  
  value = 48;  
}

Bottom of Form