**LibSX1217X**

LibSX127X este o librarie de Python3 ce faciliteaza utilizarea modulelor LoRa bazate pe chip-urile SX127X. Acesta librarie se bazeaza pe node.js si libraria SX127X de Node.js. Practic LibSX127X asigura conexiunea intre Python3 si Node.JS.

**Receptia folosind LIBSX127X:**

Pentru a receptiona un mesaj aceasta librarie trebuie sa primeasca ca si parametru o functie ce va fi apelata de fiecare dat cand se primeste un mesaj nou. Pentru a imbunatati atat numaraul de pachete ce pot fi receptionate , cat si numarul de clienti ce pot comunica cu serverul functia primita ca parametru va fi apelata intr-un nou Thread. Acest lucru presupune ca atunci cand un mesaj a fost primit treadul principal cereaza un nou thread in care va rula functia pentru receptie. Imediat dupa asta threadul pricinpal putand primi un nou mesaj pe care il pote trimite in alt thread. Putand procesa mai multe mesaje si receptiona altul in acelasasi timp

Exemplu de receptionare folosind SX127X:

from libSx127x.LoRa import LoRa

Importarea biblioteci linSx127x

lora= LoRa()

Crearea obiectului de tipul LoRa

lora.setFrequency('868e6')

Setarea Frecventei purtatoarei pentru receptie

def onRecived(message, signal):

print(message,signal)

#lora.exitReciveMode()

Definirea functiei ce va fi apelata cand se receptioneaza un nou mesaj, in cazul acesta functia afiseaza mesaju primit in linia de comanda. Aceasta functie trebuie sa aibe obligatori 2 parametri , mesaj si semnal ce vor fi setati de biblioteca in momentul apelari.

lora.setReciveMod(onRecived)

Cu ajutorul acestei comenzi se transmite biblioteci functia ce va fi apelata in momentul receptionari mesajului. Atentie aceasta comanda duce programul intr-o bucla infinita ce poate fi oprita doar folosind metoda lora.exitReciveMode() , daca este apelata din interiorul functiei transmisa ca parametru.

**Transmiterea mesajelor folosind LibSX127X:**

Pentru a transmite un mesaj biblioteca pune la dispozitie functia send(mesaj) , “mesaj” fiind o variabile te tip string ce contine date ce vor fi trimise.

Exemplu de cod utilizat pentru a transmite un mesaj:

from libSx127x.LoRa import LoRa

import time

Importarea bibliotecilor utilizate

lora= LoRa()

Crearea obiectului de tipul LoRa

lora.setFrequency('868e6')

Setarea frecventei purtatoarei pentru emisie

while True:

lora.send("Hello "+str(i))

i+=1

time.sleep(1)

Crearea unei bucle infinite in care va fi apelata metoda send a obiectului lora pentru a transmite mesajul “Hello” urmat un un numar ce se incrementeaza la fiecare transmisie. Tod in cadrul acestei bucle se foloseste si comanda time.sleep(1) pentru a limita numarul de pachete trimise la 1 pe secuda.

Classa LoRa:

Aceasta clasa apelaza 2 fisiere scriese in Node.JS(LoRa\_reciver.js, LoRa\_send.js) din lina de comanda utilizand biblioteca subroces pentru a receptiona, respectiv a transmite date.

import subprocess, time, os, signal, threading

Importarea bibliotecilor utilizate

sendFile='libSx127x/LoRa\_send.js'

reciveFile='libSx127x/LoRa\_reciver.js'

nodeAddress='node'

Definirea caii de acces catre fisierele de node.js precum si comanda utilizate pentru a le rula.

class LoRa():

Definirea clasei LoRa

def \_\_init\_\_(self):

self.freq=0

self.log=[]

self.reciveMod=False

self.loraInit=False

Definirea constructorului clasei , si a campurilor folosite

def setFrequency(self , frequency):

self.freq=frequency

Definirea funtiei pentru setarea frecventei . Aceasta seteaza campul cu numele self.freq cu valoare primita ca parametru

def send(self , message):

self.exitReciveMode()

comand = nodeAddress + " " + sendFile + " " + self.freq + " '" + message + "'"

self.sendProcess=subprocess.Popen(comand, stdout=subprocess.PIPE, shell=True)

all\_Line=[]

for i in range(0,2):

line = self.sendProcess.stdout.readline();

decodeLine=line.decode("utf-8")

decodeLine=decodeLine.replace('\n','')

all\_Line.append(decodeLine)

self.sendProcess.kill()

print(all\_Line)

Definirea funtiei send utilizata pentru a transmite date. In primul rand se opreste ori ce transmisie daca nefinalizata daca este cazul, se creaza comanda ce va fi data in linia de comanda prin concatenarea mai multor string-uri si se executa utilizand biblioteca subproces. Dupa acesti pasi se monitoriazeaza output-ul comenzi pentru a stabili daca transmisia o fost fcuta.

def setReciveMod(self, onReceiveFunction):

self.exitReciveMode()

self.reciveMode=True

comand = nodeAddress + " " + reciveFile + " " + self.freq

self.reciveProcess=subprocess.Popen(comand, stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)

allDataReceived=[]

i=0

line = self.reciveProcess.stdout.readline();

decodeLine=line.decode("utf-8")

if 'open success' in decodeLine:

self.loraInit=True

else:

self.loraInit=False

self.reciveMode=False

print("Error to conect LoRa module")

while self.reciveMode:

line = self.reciveProcess.stdout.readline();

decodeLine=line.decode("utf-8")

decodeLine=decodeLine.replace("\n",' ')

splitLine=decodeLine.split(";")

threading.Thread(name="Reciv Thread", target=onReceiveFunction, args=[splitLine[0], splitLine[1]]).start()

#onReceiveFunction(splitLine[0], splitLine[1]

Functia setRecevMode este utilizata pentru pune modulul LoRa in receptie si a apela o functie primita ca parametru in momentul primirii unui mesaj folosind un thread nou.

In primul rand se incearca oprirea altei transmisiuni daca este cazul, si se seteaza fleg-ul self.reciveMode. Se creaza comanda ce va fi data in linia de comanda prin concatenarea unor variabile de tipul string si se executa. Se astepta ca in linia de comanda sa fie afisat mesajul “open success” garantad ca a fost stabilita conexiunea cu modulul LoRa. Dupa acest pas se asteapta tiparirea in linia de comanda a mesajelor primite de catre scriptul LoRa\_reciver.js . Cand se detecteaza un mesaj se creaza un nou thread si se apeleaza din acesta functia primita ca parametru.

def exitReciveMode(self):

try:

self.reciveMode=False

os.kill(self.reciveProcess.pid, signal.SIGKILL)

except:

Pass

Cu ajutorul acestei functii se inchide modul de receptie. Se utilizeaza un bloc de tipul try , except pentru a nu opri executia programul in cazu in care se apeleaza functia fara a fi un proces de receptie activ, se seteaza flag-ul self.reciveMode=False si se opreste procesul de receptie prin comanda os.kill(self.reciveProcess.pid, signal.SIGKILL) .

**Instalarea Biblioteci SX127x**

1 Se va crea un folder nou ce va gazdui proiectul ce utilizeaza libSX127x.

2 Se descarca arhiva din link-ul de mai jos si se muta in directorul creat mai sus(https://ritual-distribution.000webhostapp.com/Alex/LoRa\_Python\_Lib.zip)

3 Se deschide un terminal si se navigheaza pana in folderul cread mai sus

4 se ruleaza comanda *cd libSx127x*

5 Se dezinstaleaza toate versiunile node.js folosind comanda sudo apt-get remove nodejs

daca nu este cazul se sare la punctul 6

6 Se instaleaza versiunea 4 ne node.js

Raspberry Pi Model A, B, B+ and Compute Module

wget https://nodejs.org/dist/v4.0.0/node-v4.0.0-linux-armv6l.tar.gz

tar -xvf node-v4.0.0-linux-armv6l.tar.gz

cd node-v4.0.0-linux-armv6l

Raspberry Pi 2 Model B

wget https://nodejs.org/dist/v4.0.0/node-v4.0.0-linux-armv7l.tar.gz

tar -xvf node-v4.0.0-linux-armv7l.tar.gz

cd node-v4.0.0-linux-armv7l

7 Se muta totul in /usr/local/ folosind comanda: sudo cp -R \* /usr/local/

8 Se testeza versiunea de node.js instalata folosind comanda node-v

9 Se instaleaza SX127X (biblioteca de node.js ) folosind comanda npm install sx127x