**SOFTWARE DESCRIPTION**

**1.TRANSMITTING DATA FROM ARDUINO TO RASPBERRY PI**

**AIM:** To send data from Arduino IDE to raspberry pi through lora sx1278.

**TOOLS REQUIRED:** ARDUINO IDE, RASPBIAN BUSTER WITH DESKTOP

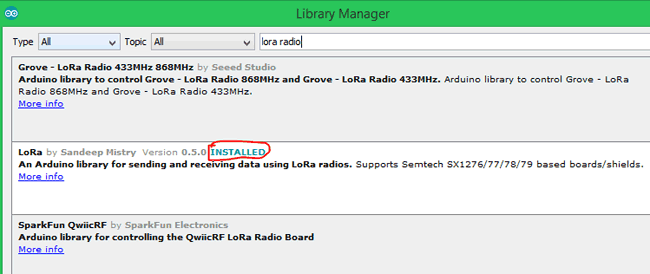
**ARDUINO 1.8.7**

Arduino IDE is an open-source software program that allows users to write and upload code within a real-time work environment. The Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. The system is fully compatible with any Arduino software board.

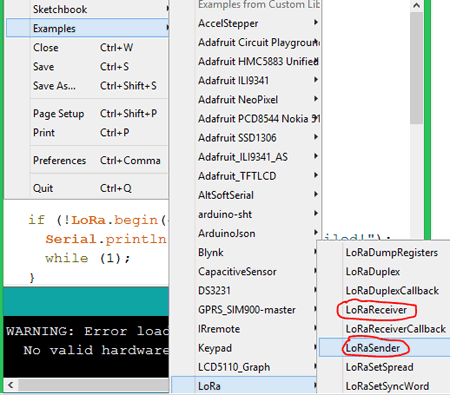
**PREPARING THE ARDUINO IDE FOR LORA WIRELESS COMMUNICATION**

Once the hardware is ready we can move on to the Arduino IDE. To work with LoRa module using Arduino, we already have a well-built [LoRa library by Sandeep Mistry](https://github.com/sandeepmistry/arduino-LoRa).  In this article we will just include the Library to our Arduino IDE and use the example sketches with slight modifications to make our LoRa modules communicate between them.

To add the library, open you Arduino IDE and follow Sketch -> Include Library -> Manage Libraries. Then search for “LoRa Radio” and look **for** the library that was made by Sandeep Mistry and click on install. Wait for the installation to complete and you should see something like this in the end



open the LoRa Example program by using **File -> Example -> LoRa** and then open LoRa Sender Program as shown below.



**RASPBIAN BUSTER WITH DESKTOP**

The Raspbian Buster released as the successor of Raspbian Stretch which was the default Raspberry Pi operating system for almost 2 years, is compatible with all the models of Raspberry Pi. Buster is based on the latest version of Debian Linux 10, with Linux kernel version 4.19, and 8.3 GCC compiler. As the previous Raspbian operating systems, Buster also provides users with the PIXEL (composed of LXDE) desktop environment.

**CODE FOR TRANSMITTING DATA FROM ARDUINO IDE**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| #include <SPI.h>  #include <LoRa.h> int sensorpin=A0; // Set the input pin for the sensor int sensorvalue; int counter = 0; void setup() {   Serial.begin(9600);   while (!Serial);   Serial.println("LoRa Sender");   if (!LoRa.begin(434E6)) // Set the frequency according to the lora module   {     Serial.println("Starting LoRa failed!");     while (1);   } }  void loop() {   sensorvalue=analogRead(A0);  // read the sensor data   Serial.println(sensorvalue);   LoRa.beginPacket();              // LoRa packets begins   LoRa.println("photoresistor");   // The data that has to be transmitted.  This gets printed at the receiver side.       LoRa.print(sensorvalue);         // The sensor value is sent to the receiver  through LoRa module   LoRa.endPacket();               // LoRa packet ends   delay(500); }  **CODE FOR RECEIVING DATA**  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()  lora.on\_rx\_done()  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)  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)  **DESCRIPTION:**  Photo resistor sensor has connected to Arduino UNO and the value of the  Sensor has been sent from Arduino IDE through lora sx1278 and data has  been visualized in Raspbian.  OUTPUT    **2.FULL DUPLEX MODE OF COMMUNICATION BETWEEN ARDUINO AND**  **RASPBERRY PI THROUGH LORA SX1278**  **AIM: To transmit and receive the data from both Arduino and raspberry pi**  **COMPONENTS REQUIRED: Arduino IDE, Raspbian buster with desktop**  **CODE FOR TRANSMITTING AND RECEIVING DATA FROM ARDUINO**  #include <SPI.h>  #include <LoRa.h>  int counter=0;  char c=1;  int b=0;  int sensorpin=A0; // Set the input pin for the sensor  int sensorvalue;  void setup() {  Serial.begin(9600);  while (!Serial);  Serial.println("LoRa Receiver");  if (!LoRa.begin(434E6)) {  Serial.println("Starting LoRa failed!");  while (1);  }  }  void loop(){  // try to parse packet  int packetSize = LoRa.parsePacket();  if (packetSize) {  // received a packet  Serial.print("Received packet");  // read packet  while (LoRa.available()) {  c=((char)LoRa.read());  Serial.print((char)LoRa.read());  if(c=='1'){  for(b=0;b<=10;b++){  sender();  }  }  }  // print RSSI of packet  //Serial.print("' with RSSI ");  //Serial.println(LoRa.packetRssi());  }  }  void sender(){  sensorvalue=analogRead(A0); // read the sensor data  Serial.println(sensorvalue);  LoRa.beginPacket(); // LoRa packets begins  LoRa.print(sensorvalue);  LoRa.endPacket();  delay(3000);  }  **CODE FOR TRANSMITTING AND RECEIVING DATA FROM**  **RASPBERRY PI**  from time import sleep  from SX127x.LoRa import\*  from SX127x.board\_config import BOARD  from urllib import request  from urllib import parse  BOARD.setup()  class LoRaRcvCont(LoRa):  tx\_counter=0  b=0  def \_\_init\_\_(self, verbose=False):  super(LoRaRcvCont, self).\_\_init\_\_(verbose)  self.set\_mode(MODE.SLEEP)  def start(self):  self.reset\_ptr\_rx()  sys.stdout.write("\rstart")  self.set\_mode(MODE.TX)  self.set\_mode(MODE.RXCONT)  self.write\_payload([0x0f])  lora.on\_tx\_done()  self.b=0  sleep(1)  while(1):  if(self.b<=5):  sleep(1)  self.b=self.b+1  rssi\_value = self.get\_rssi\_value()  status = self.get\_modem\_status()  print(self.b)  lora.on\_rx\_done()  sleep(1)  sys.stdout.flush()  elif(self.b>5):  lora.start()  def on\_rx\_done(self):  self.set\_mode(MODE.STDBY)  print("Received: ")  self.clear\_irq\_flags(RxDone=1)  payload = self.read\_payload(nocheck=True)  print(bytes(payload).decode("utf-8",'ignore'))  html=bytes(payload).decode("utf-8",'ignore')  data=request.urlopen("https://api.thingspeak.com/update?api\_key=  G11K5Q42ZJWO3R26&field1=%s"%html)  sleep(0.5)  self.set\_mode(MODE.SLEEP)  self.reset\_ptr\_rx()  self.set\_m ode(MODE.RXCONT)  def on\_tx\_done(self):  self.set\_mode(MODE.STDBY)  self.clear\_irq\_flags(TxDone=1)  sys.stdout.flush()  rawinput = str(input("Enterthenode:"))  data =[int(hex(ord(c)), 0) for c in rawinput]  self.write\_payload(data)  self.set\_mode(MODE.TX)  lora=LoRaRcvCont()  lora.set\_mode(MODE.STDBY)  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()  **DESCRIPTION:**  From raspberry we access the different Arduino by using single digit address we  compare the Arduino by calling Arduino address .If the address has been matched  Photo resistor value will be send to raspberry pi.  **OUTPUT:**    **DISPLAYING THE SENSOR DATA IN THINKSPEAK CLOUD**    **3.TRANSMITTING AND RECEIVING DATA FROM ARDUINO**  **AND RASPBERRY PI USING ANY ADDRESS**  **AIM**:  To transfer and receive data from both arduino and raspberry pi using any address.  **COMPONENTS REQUIRED**: ARDUINO IDE,RASPBIAN WITH DESKTOP  **CODE FOR TRANSMITTING AND RECEIVING DATA FROM ARDUINO**:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | #include <SPI.h> #include <LoRa.h> int sensorpin = A0; // Set the input pin for the sensor int sensorvalue;//variable to store the sensor value char c; String myString; void setup() {   Serial.begin(9600); //To begin the Serial Monitor   while (!Serial);   Serial.println("LoRa Receiver");  if (!LoRa.begin(434E6))  //to set the frequency range to recieve data   {     Serial.println("Starting LoRa failed!");     while (1);   } } void loop() {   int packetSize = LoRa.parsePacket();   if (packetSize) {     // received a packet     Serial.println("Received packet");      //to read the packet     while (LoRa.available()) {       c = ((char)LoRa.read());       String stringOne = String(c);       myString.concat(c);       Serial.print(myString);           if (myString=="12A")       {           myString.remove(0,100);           Serial.print(myString);         for (int i = 0; i <= 10; i++)         {           sender(); //sender function         }       }               Serial.print("not");     }   } } void sender() {   sensorvalue = analogRead(A0);  //to read the data from the sensor   Serial.print("Sending packet: ");    // read the sensor data   Serial.println(sensorvalue);    LoRa.beginPacket();                LoRa.print("value");      LoRa.println(sensorvalue);    LoRa.endPacket();//End Packets    myString.remove(0);    Serial.print(myString);   delay(1000); //the whole process works with an delay of 1 second }   |  |  | | --- | --- | |  |  | |  |  |   **DESCRIPTION:**  When the address is given from raspberry pi the sensor value from arduino  is transmitted to raspberry Pi   |  |  | | --- | --- | |  |  |   OUTPUT:    4.TO SEND SMS FROM MOBILE TO RPI THROUGH GSM MODULE  AND SENSOR DATA RECEIVED FROM ARDUINO TRANSMITTED  TO CLOUD THROUGH WIFI.  AIM: To transmit the sensor data from RPI to cloud through wifi .  COMPONENTS REQUIRED:Arduino Ide,Raspbian with Desktop.  **LIBRARY FOR GSM CONNECTION** |  |  |
| import os,time,sys import serial  def convert\_to\_string(buf):     try:         tt =  buf.decode('utf-8').strip()         return tt     except UnicodeError:         tmp = bytearray(buf)         for i in range(len(tmp)):             if tmp[i]>127:                 tmp[i] = ord('#')         return bytes(tmp).decode('utf-8').strip()  class SIM800L:     def \_\_init\_\_(self,ser):         try:             self.ser=serial.Serial("/dev/serial0", baudrate=9600, timeout=5)         except Exception as e:             sys.exit("Error: {}".format(e))         self.incoming\_action = None         self.no\_carrier\_action = None         self.clip\_action = None         self.\_clip = None         self.msg\_action = None         self.\_msgid = 0         self.savbuf = None      def setup(self):         self.command('AT\n')                 self.command('ATE0\n')# command echo off         self.command('AT+CLIP=1\n')    # caller line identification         self.command('AT+CMGF=1\n')    # plain text SMS         self.command('AT+CLTS=1\n')    # enable get local timestamp mode         self.command('AT+CSCLK=0\n')   # disable automatic sleep      def callback\_incoming(self,action):         self.incoming\_action = action      def callback\_no\_carrier(self,action):         self.no\_carrier\_action = action      def get\_clip(self):         return self.\_clip             def callback\_msg(self,action):         self.msg\_action = action      def get\_msgid(self):         return self.\_msgid      def command(self, cmdstr, lines=1, waitfor=500, msgtext=None):         while self.ser.in\_waiting:             self.ser.readline()         self.ser.write(cmdstr.encode())         if msgtext:             self.ser.write(msgtext.encode())         if waitfor>1000:             time.sleep((waitfor-1000)/1000)         buf=self.ser.readline() #discard linefeed etc         #print(buf)         buf=self.ser.readline()         if not buf:             return None         result = convert\_to\_string(buf)         if lines>1:             self.savbuf = ''             for i in range(lines-1):                 buf=self.ser.readline()                 if not buf:                     return result                 buf = convert\_to\_string(buf)                 if not buf == '' and not buf == 'OK':                     #self.savbuf += buf+'\n'                     self.savbuf += buf         return result      def send\_sms(self,destno,msgtext):         result = self.command('AT+CMGS="{}"\n'.format(destno),99,5000,msgtext+'\x1A')         if result and result=='>' and self.savbuf:             params = self.savbuf.split(':')             if params[0]=='+CUSD' or params[0] == '+CMGS':                 return 'OK'         return 'ERROR'      def read\_sms(self,id):         result = self.command('AT+CMGR={}\n'.format(id),99)         if result:             params=result.split(',')             if not params[0] == '':                 params2 = params[0].split(':')                 if params2[0]=='+CMGR':                     #number = params[1].replace('"',' ').strip()                     #date   = params[3].replace('"',' ').strip()                     #time   = params[4].replace('"',' ').strip()                     return(self.savbuf)         #return None         def delete\_sms(self,id):         self.command('AT+CMGD={}\n'.format(id),1)      def check\_incoming(self):         if self.ser.in\_waiting:             buf=self.ser.readline()             # print(buf)             buf = convert\_to\_string(buf)             params=buf.split(',')              if params[0][0:5] == "+CMTI":                 self.\_msgid = int(params[1])                 if self.msg\_action:                     self.msg\_action()              elif params[0] == "NO CARRIER":                     self.no\_carrier\_action()              elif params[0] == "RING" or params[0][0:5] == "+CLIP":                 #@todo handle                 pass      def read\_and\_delete\_all(self):         try:             return self.read\_sms(1)         finally:             self.command('AT+QMGDA="DEL ALL"\n',1)  sim=SIM800L("/dev/serial0") sim.setup(  **CODE TO SEND MESSAGE FROM RASPBERRY PI TO ARDUINO IDE**  **& RECEIVED DATA PUSHED TO CLOUD THROUGH WIFI**   |  |  |  | | --- | --- | --- | |  | from time import sleep from gsm\_1 import sim # GSM library is imported to the code. from SX127x.LoRa import\* from SX127x.board\_config import BOARD import json import requests from datetime import datetime BOARD.setup() class LoRaRcvCont(LoRa):     tx\_counter=0     b=0     html=None     url = '<http://infinitesol.info/irriot/api/operation/Syncstatus>'     def \_\_init\_\_(self, verbose=False):        super(LoRaRcvCont, self).\_\_init\_\_(verbose)         self.set\_mode(MODE.SLEEP)       def start(self):         self.reset\_ptr\_rx()         sys.stdout.write("\rstart")         self.set\_mode(MODE.TX)         self.set\_mode(MODE.RXCONT)         self.write\_payload([0x0f])         sim.read\_and\_delete\_all()         sim.send\_sms("9677534206","waiting to receive")         lora.receive\_sms()         #lora.on\_tx\_done()         self.b=1         sleep(1)         while(1):             if(self.b<=5):                 sleep(1)                 self.b=self.b+1                 rssi\_value = self.get\_rssi\_value()                 status = self.get\_modem\_status()                 print(self.b)                 lora.on\_rx\_done()                 sleep(1)                                 sys.stdout.flush()             elif(self.b>5):                                 lora.start()                             def on\_rx\_done(self):         self.set\_mode(MODE.STDBY)         print("Received: ")         self.clear\_irq\_flags(RxDone=1)         payload = self.read\_payload(nocheck=True)         print(bytes(payload).decode("utf-8",'ignore'))         self.html=(bytes(payload).decode("utf-8",'ignore'))         #sleep(1.5)         sleep(2)         lora.cloud()         self.set\_mode(MODE.SLEEP)         self.reset\_ptr\_rx()         self.set\_mode(MODE.RXCONT)  def on\_tx\_done(self):         #global args         self.set\_mode(MODE.STDBY)         self.clear\_irq\_flags(TxDone=1)         sys.stdout.flush()         print("Waitng to receive  the node...")         sleep(5)         print(self.node2)             data =[int(hex(ord(c)), 0) for c in self.node2]         self.write\_payload(data)         self.set\_mode(MODE.TX)             def receive\_sms(self):         print("waiting...")                 self.node2=sim.read\_sms(1)         #print(self.node2)         if(self.node2==None):             lora.receive\_sms()         else:                         lora.on\_tx\_done()  def cloud(self):         e=datetime.now()         now=str(e)         myobj = {           "accId": 1,           "userId": 1,           "statusdata":[           {             "statusdate": now,             "phoneno": "+918148425113",             "message": self.html           }         ]         }         headers={'content-type':'application/json','accept':'text/plain'}         n=json.dumps(myobj,indent=2)         x = requests.post(self.url, data=n,headers=headers)         print(x.text)  lora=LoRaRcvCont() lora.set\_mode(MODE.STDBY) lora.set\_pa\_config(pa\_select=1) sim.read\_and\_delete\_all() print("Welcome") sim.send\_sms("+919677534206","system starts") print("Waitng to receive...") sleep(5) def read\_sms1():     node=sim.read\_sms(1)     print(node)     if(node=='1'):                 lora.start()  elif(node==None):         read\_sms1()     else:         read\_sms1() try:     node=sim.read\_sms(1)     print(node)     if(node=='1'):         lora.start()  elif(node==None):         read\_sms1()   except KeyboardInterrupt:     sys.stdout.flush()     print("")     sys.stderr.write("KeyboardInterrupt\n") finally:     sys.stdout.flush()     print("")     lora.set\_mode(MODE.SLEEP)     BOARD.teardown()  **CODE FOR TRANSMITTING THE SENSOR VALUE FROM ARDUINO TO**  **RASPBERRY PI AFTER RECEIVING DATA FROM RPI**  #include <SPI.h> #include <LoRa.h> int sensorpin = A0; // Set the I nput pin for the sensor int sensorvalue;//variable to store the sensor value char c; String myString; int LED = 5; void setup() {   Serial.begin(9600); //To begin the Serial Monitor   //while (!Serial);   Serial.println("LoRa Receiver");   if (!LoRa.begin(434E6))  //to set the frequency range to recieve data   {     Serial.println("Starting LoRa failed!");     while (1);   } } void loop() {   // try to parse packet   int packetSize = LoRa.parsePacket();   if (packetSize) {     // received a packet     Serial.println("Received packet");      //to read the packet     while (LoRa.available()) {       c = ((char)LoRa.read());       myString.concat(c);       Serial.print(myString);     }     if (myString == "VL1 ON")     {       digitalWrite(LED, HIGH);       for (int i = 0; i <= 6; i++)       {         sensorvalue = analogRead(A0);//to read the data from the sensor                 Serial.print(myString);         delay(4000);         Serial.print("CID:1;S1:");         Serial.print(sensorvalue);         Serial.print(";V1:0ON");         Serial.println("");         LoRa.beginPacket();         LoRa.print("CID:1;S1:");         LoRa.print(sensorvalue);         LoRa.print(";V1:0ON");         LoRa.endPacket();//End Packets         //Serial.println("VALVE1 ON");         myString.remove(0, 100);       }     }     else if (myString == "VL1 OFF")     {        digitalWrite(LED, LOW);       for (int i = 0; i <= 6; i++)       {                 sensorvalue = analogRead(A0);//to read the data from the sensor         Serial.print(myString);         delay(4000);         Serial.print("CID:1;S1:");         Serial.print(sensorvalue);         Serial.print(";V1:OFF");         Serial.print("");         LoRa.beginPacket();         LoRa.print("CID:1;S1:");         LoRa.print(sensorvalue);         LoRa.print(";V1:OFF");         LoRa.endPacket();//End Packets         //Serial.println("VALVE OFF");         myString.remove(0, 100);       }     }     else     {       myString.remove(0, 100);       Serial.print("not");     }    } }  **EXPLANATION**:  When the sms came from the mobile to turn on the valve RPI sends the data to arduino and  Arduino turns on the relay and sensor data sent to RPI and RPI sends the data to cloud through wifi.  **5. TO SEND SMS FROM MOBILE TO RPI THROUGH GSM MODULE**  **AND SENSOR DATA RECEIVED FROM ARDUINO TRANSMITTED**  **TO CLOUD THROUGH GPRS CONNECTION**  **AIM**: To transmit the sensor data from RPI to cloud through GPRS connection .  **COMPONENTS REQUIRED**:Arduino Ide,Raspbian with Desktop.  **CODE TO SEND MESSAGE FROM RASPBERRY PI TO ARDUINO IDE &**  **RECEIVED DATA PUSHED TO CLOUD THROUGH GPRS CONNECTION** |  |  |  |  | | --- | --- | |  |  |   from time import sleep  from gprs import sim  from SX127x.LoRa import\*  from SX127x.board\_config import BOARD  import json  import requests  from datetime import datetime  BOARD.setup()  class LoRaRcvCont(LoRa):    tx\_counter=0  b=0  html=None  url='GET /~mayagree/api.php?MSG=CID:1;S2:65;V2:0ON\r\n'  d=[]    def \_\_init\_\_(self, verbose=False):    super(LoRaRcvCont, self).\_\_init\_\_(verbose)  self.set\_mode(MODE.SLEEP)    def start(self):    self.reset\_ptr\_rx()  sys.stdout.write("\rstart")  self.set\_mode(MODE.TX)  self.set\_mode(MODE.RXCONT)  self.write\_payload([0x0f])  sim.read\_and\_delete\_all()  self.d.clear()  sim.send\_sms("9677534206","waiting to receive")  lora.receive\_sms()  #lora.on\_tx\_done()  self.b=1  sleep(1)  while(1):  if(self.b<=6):  sleep(1)  self.b=self.b+1  rssi\_value = self.get\_rssi\_value()  status = self.get\_modem\_status()  lora.on\_rx\_done()  sleep(1)  sys.stdout.flush()    elif(self.b==7):  print(self.d)  #continue  lora.gprs()      def on\_rx\_done(self):  self.set\_mode(MODE.STDBY)  print("Received: ")  self.clear\_irq\_flags(RxDone=1)  payload = self.read\_payload(nocheck=True)  print(bytes(payload).decode("utf-8",'ignore'))  self.html=(bytes(payload).decode("utf-8",'ignore'))  #print(type(self.html))  self.d.append(self.html)  sleep(1.5)  #sleep(2)  #lora.gprs()  #sim.gprs\_gsm(length,data)  #lora.cloud()  self.set\_mode(MODE.SLEEP)  self.reset\_ptr\_rx()  self.set\_mode(MODE.RXCONT)    def on\_tx\_done(self):  #global args  self.set\_mode(MODE.STDBY)  self.clear\_irq\_flags(TxDone=1)  sys.stdout.flush()  print("Waitng to receive the node...")  sleep(5)  print(self.node2)  data =[int(hex(ord(c)), 0) for c in self.node2]  self.write\_payload(data)  self.set\_mode(MODE.TX)    def receive\_sms(self):  print("waiting...")  self.node2=sim.read\_sms(1)  #print(self.node2)  if(self.node2==None):  lora.receive\_sms()  else:  lora.on\_tx\_done()  def cloud(self):  e=datetime.now()  now=str(e)  myobj = {  "accId": 1,  "userId": 1,  "statusdata":[  {  "statusdate": now,  "phoneno": "+918148425113",  "message": self.html  }  ]  }  headers={'content-type':'application/json','accept':'text/plain'}  n=json.dumps(myobj,indent=2)  x = requests.post(self.url, data=n,headers=headers)  print(x.text)  def gprs(self):  data=('GET /~mayagree/api.php?MSG='+self.d[6]+'\r\n')  length=str(len(data))  print(data)  sim.gprs\_gsm(length,data)  sleep(5)  lora.start()  **EXPLANATION:**  When the sms came from the mobile to turn on the valve then RPI receives the message from app  through gsm module & sends the received message to arduino through lora and Arduino turns on the  relay and sensor data sent to RPI and RPI sends the data to cloud through GPRS connection.  OUTPUT:    SENSOR DATA SENT TO CLOUD THROUGH GPRS CONNECTION  **6. TO SET CYCLIC TIMER & GROUPING OF VALVES**  **AIM:**To set cyclic timer & grouping of valves using mobile application.  **COMPONENTS REQUIRED:ARDUINO IDE,RASPBIAN WITH DESKTOP**  **CODE TO SET THE CYCLIC TIMER & GROUPING OF VALVES IN RPI**  from time import sleep from gprs import sim from SX127x.LoRa import\* from SX127x.board\_config import BOARD import json import requests from datetime import datetime  BOARD.setup()  class LoRaRcvCont(LoRa):         tx\_counter=0     b=0     r=None     grp=None     group\_read=None     sep\_group=None     G=[]     GP=[]     start\_time=[]     end\_time=[]     group=[]     d=[]     ln=None  def \_\_init\_\_(self, verbose=False):         super(LoRaRcvCont, self).\_\_init\_\_(verbose)         self.set\_mode(MODE.SLEEP)               def start(self):  self.reset\_ptr\_rx()         sys.stdout.write("\rstart")         self.set\_mode(MODE.TX)         self.set\_mode(MODE.RXCONT)         self.write\_payload([0x0f])         sim.read\_and\_delete\_all()         self.d.clear()         sim.send\_sms("+919677534206","system starts")         lora.receive\_sms()         self.b=1         sleep(1)             def receiver(self):         self.b=1         sleep(1)         while(1):             if(self.b<=5):                 sleep(1)                 self.b=self.b+1                 rssi\_value = self.get\_rssi\_value()                 status = self.get\_modem\_status()                 print(self.b)                 lora.on\_rx\_done()                 sleep(1)                                 sys.stdout.flush()             elif(self.b==6):                                 lora.gprs()                 def on\_rx\_done(self):         self.set\_mode(MODE.STDBY)         print("Received: ")         self.clear\_irq\_flags(RxDone=1)         payload = self.read\_payload(nocheck=True)         print(bytes(payload).decode("utf-8",'ignore'))         self.html=(bytes(payload).decode("utf-8",'ignore'))         a=len(self.html)         if(a>=15):             self.d.append(a)             sleep(2)         self.set\_mode(MODE.SLEEP)         self.reset\_ptr\_rx()         self.set\_mode(MODE.RXCONT)  def on\_tx\_done(self):  #global args         self.set\_mode(MODE.STDBY)         self.clear\_irq\_flags(TxDone=1)         sys.stdout.flush()         print("Waitng to receive  the node...")         sleep(2)         print(self.node2)         data =[int(hex(ord(c)), 0) for c in self.node2]         self.write\_payload(data)         self.set\_mode(MODE.TX)         lora.receiver()             def receive\_sms(self):         print("waiting...")                 self.node2=sim.read\_sms(1)         if(self.node2==None):             lora.receive\_sms()         elif(self.node2=="AUTO ON"):             print("cyclic timer")             sim.read\_and\_delete\_all()             lora.cyclic\_timer()         else:                         lora.on\_tx\_done()  def gprs(self):         a=len(self.d)         if(a==0):             sim.send\_sms("9677534206","Interrupt")             lora.start()       else:           for i in self.d:                 print(i)                 data=('GET /~mayagree/api.php?MSG='+i+'\r\n')                 length=str(len(data))                 print(data)                 sim.gprs\_gsm(length,data)                                     lora.start()                 break             print("Interrupt")             lora.start()               def cyclic\_timer(self):         print("waiting...")         self.group\_read=sim.read\_sms(1)         if(self.group\_read==None):             lora.cyclic\_timer()         else:             lora.cyclic()                 def cyclic(self):       a=self.group\_read         sim.read\_and\_delete\_all()         b=a.split(';')         c=b[2:]                 self.ln=len(c)         print(self.ln)                 for i in range(self.ln):             d=c[i].split(':')                         e=d[1].split(',')             self.GP.append(d[0])                         self.G.append(e)             print(self.G)             print(self.GP)         lora.separate\_grp\_recevier()      def separate\_grp\_recevier(self):         print("waiting...")         self.sep\_group=sim.read\_sms(1)         if(self.sep\_group==None):             lora.separate\_grp\_recevier()         else:             lora.separate\_grp()                 def separate\_grp(self):             self.grp=self.sep\_group         print(self.grp)         sim.read\_and\_delete\_all()         if(len(self.grp)>=20):             lora.timer()                         def check():                 global i                 for i in range(self.ln):                     CID={i:{self.GP[i]:self.G[i],self.start\_time[i]:self.end\_time[i]}}                     qw=self.GP[i]                     if(rn==self.start\_time[i]):                         for i in self.G[i]:                             d=str(qw+":"+i+" ON")                             self.set\_mode(MODE.STDBY)                             self.clear\_irq\_flags(TxDone=1)                             sys.stdout.flush()                             print("Waitng to receive  the node...")                             sleep(12)                             print(d)                             data =[int(hex(ord(c)), 0) for c in d]                             self.write\_payload(data)                             self.set\_mode(MODE.TX)                                      elif(rn==self.end\_time[i]):                         for i in self.G[i]:                             e=str(qw+":"+i+" OFF")                             self.set\_mode(MODE.STDBY)                             self.clear\_irq\_flags(TxDone=1)                             sys.stdout.flush()                             print("Waitng to receive  the node...")                             sleep(12)                             print(e)                             data =[int(hex(ord(c)), 0) for c in e ]                             self.write\_payload(data)                             self.set\_mode(MODE.TX)                                while(1):                 rn = str(datetime.now().strftime("%H:%M"))                 print(rn)                 msg=sim.read\_sms(1)                 if(msg=="AUTO OFF"):                     lora.start()                 else:                     check()                      def timer(self):             a=self.grp             b=a.split(';')             c=b[1:]             print(self.ln)             for i in range(self.ln):                 q=c[i]                 d=q.split('|')                 self.group.append(d[0])                 e=d[1]                 f=e.split('~')                                 self.start\_time.append(f[0])                 print(self.start\_time)                 self.end\_time.append(f[1])                 print(self.end\_time)                 print(self.group)             for i in range(self.ln):                 CID={self.group[i]:{self.start\_time[i]:self.end\_time[i]}}                 print(CID)      lora=LoRaRcvCont() lora.set\_mode(MODE.STDBY) lora.set\_pa\_config(pa\_select=1) sim.read\_and\_delete\_all() sleep(5) 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()  **CODE TO SET THE CYCLIC TIMER & GROUPING OF VALVES IN ARDUINO**  #include <SPI.h> #include <LoRa.h> #include "dht.h" #define dht\_apin A0 int sensorvalue; char c; String myString; int valve1 = 5; dht DHT; void setup() {   Serial.begin(9600);   Serial.println("LoRa Receiver");   if (!LoRa.begin(434E6))   {     Serial.println("Starting LoRa failed!");     while (1);   } } void loop() {   // try to parse packet   int packetSize = LoRa.parsePacket();   if (packetSize) {     // received a packet     Serial.println("Received packet");      //to read the packet     while (LoRa.available()) {       c = ((char)LoRa.read());       myString.concat(c);       Serial.print(myString);     }     Serial.println(myString);     if (myString == "VL1 ON")     {       digitalWrite(valve1, HIGH);       for (int i = 0; i <= 4; i++)       {         DHT.read11(dht\_apin);         /\*  Serial.print("Current humidity = ");              Serial.print(DHT.humidity);              Serial.print("%  ");              Serial.print("temperature = ");              Serial.print(DHT.temperature);              Serial.println("C  ");              Serial.print(myString);\*/         delay(3000);         Serial.print("CID:1;S1:");         Serial.print(DHT.humidity);         Serial.println(";V1:0ON");         Serial.print("CID:1;S2:");         Serial.print(DHT.temperature);         Serial.println(";V1:0ON");         Serial.println("");         LoRa.beginPacket();         LoRa.print("CID:1;S1:");         LoRa.print(DHT.humidity);         LoRa.println(";V1:0ON");         // LoRa.print("CID:1;S2:");         // LoRa.print(DHT.temperature);         //  LoRa.println(";V1:0ON");         LoRa.endPacket();         myString.remove(0, 100);       }     }     else if (myString == "VL1 OFF")     {        digitalWrite(valve1, LOW);       for (int i = 0; i <= 4; i++)       {         DHT.read11(dht\_apin);         Serial.print("Current humidity = ");         Serial.print(DHT.humidity);         Serial.print("%  ");         Serial.print("temperature = ");         Serial.print(DHT.temperature);         Serial.println("C  ");         Serial.print(myString);         delay(3000);         Serial.print("CID:1;S1:");         Serial.print(DHT.humidity);         Serial.println(";V1:0OFF");         Serial.print("CID:1;S2:");         Serial.print(DHT.temperature);         Serial.println(";V1:0OFF");         LoRa.beginPacket();         LoRa.print("CID:1;S1:");         LoRa.print(DHT.humidity);         LoRa.print(";V1:OFF");         //LoRa.print("CID:1;S2:");         //LoRa.print(DHT.temperature);         //LoRa.println(";V1:OFF");         LoRa.endPacket();         myString.remove(0, 100);       }     }     else if (myString == "G1:VL1 ON")     {       digitalWrite(valve1, LOW);       Serial.println("VALVE ON");        for (int i = 0; i <= 4; i++)         {         LoRa.beginPacket();         LoRa.print("V1 ON");         LoRa.endPacket();         }     }     else if (myString == "G1:VL1 OFF")     {       Serial.println("VALVE OFF");       digitalWrite(valve1, HIGH);       for (int i = 0; i <= 4; i++)         {         LoRa.beginPacket();         LoRa.print("V1 OFF");         LoRa.endPacket();         }     }      else     {        myString.remove(0, 100);       Serial.print("not");     }    } }  **EXPLANATION:**  When the message received from mobile application to automatic on & off the cyclic timer is activated ,then  grouping of valves message sent from mobile with timer.RPI receives the message & sends the message to  each node(arduino) to turn on/off the valves.  **OUTPUT:**    **CYCLIC TIMER & GROUPING OF VALVES SETTING** |  |  |