6

**LORA code for Transmitting Side(with sensor)**

#include <LoRa.h>

#include <SPI.h>

#include <dht.h>

#define dht\_apin A0 // Analog Pin sensor is connected to

dht DHT;

int count = 0;

int counter = 0;

void setup() {

delay(1000);

// Ask for firmware version

Serial.begin(9600);

while (!Serial);

Serial.println("LoRa Sender");

if (!LoRa.begin(915E6)) { /\*LoRa.begin(frequency)\*/

Serial.println("Starting LoRa failed!");

while (1);

}else{

LoRa.setTxPower(13);

LoRa.setSignalBandwidth(125E3);

LoRa.setCodingRate4(5);

}

}

void loop() {

DHT.read11(dht\_apin);

// StaticJsonDocument<200> doc;

String data = "";

count++;

data = data + "test"+(String)count + ", ";

data = data + "temp:"+(String)DHT.temperature + ", ";

data = data + "hum:"+(String)DHT.humidity + ", ";

data = data + "soil:"+(String)analogRead(1) + ", ";

Serial.print("Sending packet: ");

Serial.println(counter);

Serial.println(analogRead(1));

Serial.print("!");

Serial.print(data);

Serial.print("\r\n");

// send packet

LoRa.beginPacket();

LoRa.print(counter);

LoRa.print("\n");

LoRa.print(data);

LoRa.endPacket();

counter++;

delay(5000);

}

**LORA code for Transmitting Side**

#include <SPI.h>

#include <LoRa.h>

int counter = 0;

void setup() {

Serial.begin(9600);

while (!Serial);

Serial.println("LoRa Sender");

if (!LoRa.begin(915E6)) { /\*LoRa.begin(frequency)\*/

Serial.println("Starting LoRa failed!");

while (1);

}else{

LoRa.setTxPower(13);

LoRa.setSignalBandwidth(125E3);

LoRa.setCodingRate4(5);

}

}

void loop() {

Serial.print("Sending packet: ");

Serial.println(counter);

// send packet

LoRa.beginPacket();

LoRa.print("hello ");

LoRa.print(counter);

LoRa.endPacket();

counter++;

delay(5000);

}

**LORA code for Receiver Side**

#include <SPI.h>

#include <LoRa.h>

/\*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. \*/

void setup() {

Serial.begin(9600);

while (!Serial);

Serial.println("LoRa Receiver");

if (!LoRa.begin(915E6)) { /\*LoRa.begin(frequency)\*/

Serial.println("Starting LoRa failed!");

while (1);

}else{

LoRa.setTxPower(13);

LoRa.setSignalBandwidth(125E3);

LoRa.setCodingRate4(5);

}

}

void loop() {

// try to parse packet

int packetSize = LoRa.parsePacket();

if (packetSize) {

// received a packet

Serial.print("Received packet '");

// read packet

while (LoRa.available()) {

Serial.print((char)LoRa.read());

}

// print RSSI of packet

Serial.print("' with RSSI ");

Serial.println(LoRa.packetRssi());

}

}

<https://github.com/rpp0/gr-lora/wiki/Capturing-LoRa-signals-using-an-RTL-SDR-device>

<http://wiki.dragino.com/index.php?title=Lora/GPS_Shield#Example1_--_Getting_GPS_to_work_on_an_Arduino>

<https://www.instructables.com/id/Dragino-LoRa-GPS-Tracker-1/>

<https://www.loraserver.io/loraserver/overview/downloads/>

<https://www.loraserver.io/lora-gateway-os/overview/>

915mhz/

spreading factor to 7 -12/

bandwidth 125k 125000 /

coding rate denominator should be 5

/ \*\*\*\*\*\*\*the power level 13(the lowest) 15이상으로 하지말기

/ 40mw of power .......2-300m

#include <lmic.h>

#include <dht.h>

#include <hal/hal.h>

#include <SPI.h>

dht DHT;

#define DHT11\_PIN A0

#define PIN\_A A0

float temperature,humidity;

float tem,hum;

unsigned int count = 1; //For times count

String datastring1="";

String datastring2="";

String datastring3="";

static uint8\_t mydata[11] = {0x01,0x67,0x00,0x00,0x02,0x68,0x00,0x03,0x65,0x00,0x00};

/\* LoRaWAN NwkSKey, network session key

This is the default Semtech key, which is used by the prototype TTN

network initially.

ttn\*/

static const PROGMEM u1\_t NWKSKEY[16] = { 0xD1, 0x23, 0x66, 0x56, 0x3F, 0xC8, 0xAB, 0x49, 0xBD, 0x59, 0xD0, 0xD1, 0xC3, 0xEE, 0x08, 0xCA };

/\* LoRaWAN AppSKey, application session key

This is the default Semtech key, which is used by the prototype TTN

network initially.

ttn\*/

static const u1\_t PROGMEM APPSKEY[16] = {0x27, 0x35, 0x14, 0x64, 0x2B, 0xC0, 0x6F, 0x11, 0x49, 0x9A, 0x80, 0x7C, 0x37, 0x33, 0xEA, 0xCD };

/\*

LoRaWAN end-device address (DevAddr)

See http://thethingsnetwork.org/wiki/AddressSpace

ttn\*/

static const u4\_t DEVADDR = 0x26021C6F;

/\* These callbacks are only used in over-the-air activation, so they are

left empty here (we cannot leave them out completely unless

DISABLE\_JOIN is set in config.h, otherwise the linker will complain).\*/

void os\_getArtEui (u1\_t\* buf) { }

void os\_getDevEui (u1\_t\* buf) { }

void os\_getDevKey (u1\_t\* buf) { }

static osjob\_t initjob,sendjob,blinkjob;

/\* Schedule TX every this many seconds (might become longer due to duty

cycle limitations).\*/

const unsigned TX\_INTERVAL = 10;

// Pin mapping

const lmic\_pinmap lmic\_pins = {

.nss = 10,

.rxtx = LMIC\_UNUSED\_PIN,

.rst = 9,

.dio = {2, 6, 7},

};

void do\_send(osjob\_t\* j){

// Check if there is not a current TX/RX job running

if (LMIC.opmode & OP\_TXRXPEND) {

Serial.println("OP\_TXRXPEND, not sending");

} else {

dhtTem();

//light();

// Prepare upstream data transmission at the next possible time.

// LMIC\_setTxData2(1,datasend,sizeof(datasend)-1,0);

LMIC\_setTxData2(1, mydata, sizeof(mydata), 0);

Serial.println("Packet queued");

Serial.print("LMIC.freq:");

Serial.println(LMIC.freq);

Serial.println("Receive data:");

}

// Next TX is scheduled after TX\_COMPLETE event.

}

void onEvent (ev\_t ev) {

Serial.print(os\_getTime());

Serial.print(": ");

Serial.println(ev);

switch(ev) {

case EV\_SCAN\_TIMEOUT:

Serial.println(F("EV\_SCAN\_TIMEOUT"));

break;

case EV\_BEACON\_FOUND:

Serial.println(F("EV\_BEACON\_FOUND"));

break;

case EV\_BEACON\_MISSED:

Serial.println(F("EV\_BEACON\_MISSED"));

break;

case EV\_BEACON\_TRACKED:

Serial.println(F("EV\_BEACON\_TRACKED"));

break;

case EV\_JOINING:

Serial.println(F("EV\_JOINING"));

break;

case EV\_JOINED:

Serial.println(F("EV\_JOINED"));

break;

case EV\_RFU1:

Serial.println(F("EV\_RFU1"));

break;

case EV\_JOIN\_FAILED:

Serial.println(F("EV\_JOIN\_FAILED"));

break;

case EV\_REJOIN\_FAILED:

Serial.println(F("EV\_REJOIN\_FAILED"));

break;

case EV\_TXCOMPLETE:

Serial.println(F("EV\_TXCOMPLETE (includes waiting for RX windows)"));

if(LMIC.dataLen) {

// data received in rx slot after tx

Serial.print(F("Data Received: "));

Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);

Serial.println();

}

// Schedule next transmission

os\_setTimedCallback(&sendjob, os\_getTime()+sec2osticks(TX\_INTERVAL), do\_send);

break;

case EV\_LOST\_TSYNC:

Serial.println(F("EV\_LOST\_TSYNC"));

break;

case EV\_RESET:

Serial.println(F("EV\_RESET"));

break;

case EV\_RXCOMPLETE:

// data received in ping slot

Serial.println(F("EV\_RXCOMPLETE"));

break;

case EV\_LINK\_DEAD:

Serial.println(F("EV\_LINK\_DEAD"));

break;

case EV\_LINK\_ALIVE:

Serial.println(F("EV\_LINK\_ALIVE"));

break;

default:

Serial.println(F("Unknown event"));

break;

}

}

void setup() {

// initialize digital pin as an output.

Serial.begin(9600);

while(!Serial);

Serial.println("Connect to TTN and Send data to mydevice(Use DHT11 Sensor):");

#ifdef VCC\_ENABLE

// For Pinoccio Scout boards

pinMode(VCC\_ENABLE, OUTPUT);

digitalWrite(VCC\_ENABLE, HIGH);

delay(1000);

#endif

// LMIC init

os\_init();

// Reset the MAC state. Session and pending data transfers will be discarded.

LMIC\_reset();

//LMIC\_setupChannel(0, 902300000, DR\_RANGE\_MAP(DR\_SF7, DR\_SF7), BAND\_CENTI); // g-band

int channel = 0;

for(int i=0; i<71; i++) { // For EU; for US use i<71

if(i != channel) {

LMIC\_disableChannel(i);

}

}

/\*LMIC\_setClockError(MAX\_CLOCK\_ERROR \* 1/100);

Set static session parameters. Instead of dynamically establishing a session

by joining the network, precomputed session parameters are be provided.\*/

#ifdef PROGMEM

/\* On AVR, these values are stored in flash and only copied to RAM

once. Copy them to a temporary buffer here, LMIC\_setSession will

copy them into a buffer of its own again.\*/

uint8\_t appskey[sizeof(APPSKEY)];

uint8\_t nwkskey[sizeof(NWKSKEY)];

memcpy\_P(appskey, APPSKEY, sizeof(APPSKEY));

memcpy\_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));

LMIC\_setSession (0x1, DEVADDR, nwkskey, appskey);

#else

// If not running an AVR with PROGMEM, just use the arrays directly

LMIC\_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);

#endif

// Disable link check validation

LMIC\_setLinkCheckMode(0);

// TTN uses SF9 for its RX2 window.

LMIC.dn2Dr = DR\_SF7;

// Set data rate and transmit power (note: txpow seems to be ignored by the library)

LMIC\_setDrTxpow(DR\_SF7,14);

// Start job

do\_send(&sendjob);

}

void dhtTem()

{

int16\_t tem1;

temperature = DHT.read11(DHT11\_PIN); //Temperature detection

tem = DHT.temperature\*1.0;

float humidity = DHT.read11(DHT11\_PIN);

float hum = DHT.humidity\* 1.0;

Serial.print(F("########### "));

Serial.print(F("NO."));

Serial.print(count);

Serial.println(F(" ###########"));

Serial.println(F("The temperautre and humidity :"));

Serial.print(F("["));

Serial.print(tem);

Serial.print(F("℃"));

Serial.print(F(","));

Serial.print(hum);

Serial.print(F("%"));

Serial.print(F("]"));

Serial.println("");

count++;

tem1=(tem\*10);

mydata[2] = tem1>>8;

mydata[3]= tem1;

mydata[6] = hum \* 2;

}

void light(){

int16\_t lux;

int val,val1;

val=analogRead(PIN\_A);

Serial.print(F("a:"));

Serial.println(val);

delay(500);

val1=val\*1.0;

lux=val1;

mydata[9]=lux>>8;

mydata[10]=lux;

Serial.print(lux);

}

void loop() {

os\_runloop\_once();

}