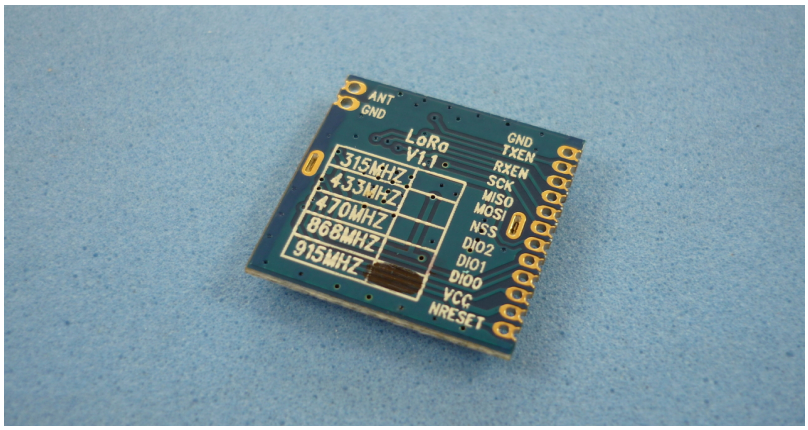
INTERNET DE LAS COSAS
LARGA DISTANCIA, BAJA POTENCIA



LPWAN y LORA

El internet de las cosas (IoT en Ingles) ha ido ganando terreno, de forma tal que cada vez es mas barato, mas pequeño y consume menos energia
comunicar "cosas" que antes eran impensables. En la actualidad tecnologias como GSM y WiFi son las mas utilizadas, estas tienen su desventajas:

- GSM, 3G, LTE:Alto costo: Requiere pago de un plan de datos para realizar la comunicacion, Alto consumo de energia.

- WiFi, BLUETOOTH : Bajo radio de cobertura (algunos cuantos metros dentro de edificaciones Alto consumo de energia!

Para mitigar las falencias que tienen las tecnologias anteriores se ha propuesto el esquema LPWAN cuyas siglas en Ingles significan
Redes de Area Amplia de Baja Potencia. para lo cual se deben cumplir ciertos requerimientos:

- Orientado a dispositivos que se comunican con poca frecuencia y volumenes de datos relativamente pequeños

- Bajo consumo de potencia, para que aparatos alimentados con baterias puedan funcionar por varios años

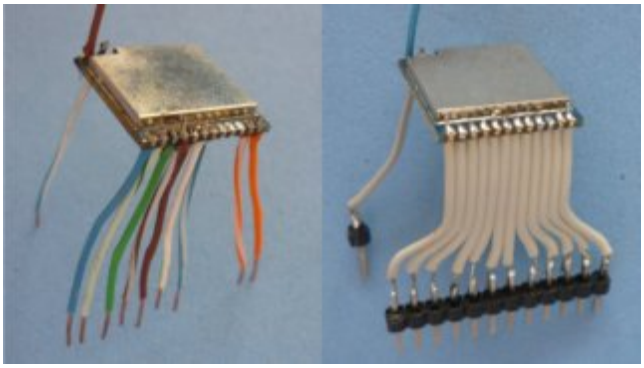
- Area amplia de cubrimiento, que un dispositivo pueda enviar y recibir datos sin problemas aun cuando se encuentre dentro de una vivienda, gabinete industrial, etc

En la actualidad para cubrir esta demanda se estan desarrollando varias tecnologias como LoRa, LTE-MTC, RPMA, UNB, Weightless entre otros.

La alianza LoRa de la cual hacen parte empresas como CISCO, IBM, SEMTECH, MICROCHIP y otras busca crear estandares de comunicacion inalambrica para dispositivos que funcionen con baterias y coberturas regional, nacional o global. La principal ventaja de LoRa es que trabaja en bandas no licenciadas ISM (Industrial, Scientific and Medical), por lo cual cualquier persona podria crear su propia red LPWAN sin tener que pagar derechos por el espectro, siempre y cuando este dentro de las reglamentaciones propias de cada pais con respecto a esta banda.

NiceRF LoRa1276

En la actualidad existen multiples fabricantes que producen modulos compatibles con LoRa. Todos ellos hacen uso de los chips producidos por SEMTECH, lo que garantiza una buena interoperabilidad entre modulos de diversos fabricantes. El modulo utilizado es el LoRa 1276, fabricado por NiceRF. Estos modulos cuestan alrededor de 20 dolares el par (envio incluido) e incorporan el chip SX1276. El fabricante indica que pueden tener un alcance maximo de 10 Km en linea de vista, y de 1Km, en ambientes urbanos, con una potencia maxima de transmision de 120 mW. Estos modulos vienen en un empaque muy poco amigable para la experimentacion, pues la distancia entre los pines (1.27 mm), no es compatible con un protoboard estandar (2.54 mm) , sin embargo con un poco de creatividad se puede hacer una adaptacion.

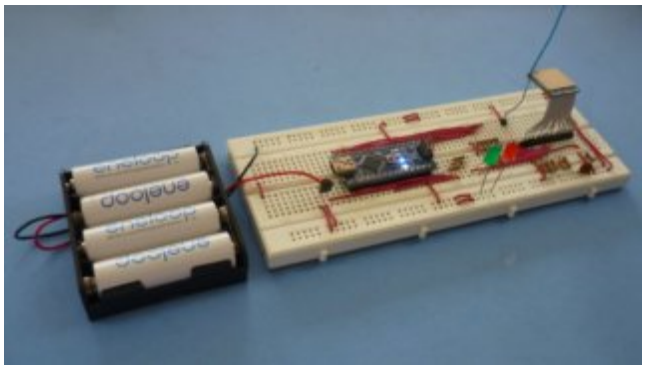


El chip que incorpora el modulo (SX1276) esta en la capacidad de operar en un rango de frecuencias de 100 a 1050 mhz. Sin embargo este chip requiere algunos componentes externos entre la salida y entrada de radio frecuencia y la antena, para minimizar la complejidad del diseño y por lo tanto costos, el fabricante del modulo pone algunos componentes como condensadores y bobinas en una red de impedancias acopladas con la antena, con lo que el modulo solo transmitira y recibira eficazmente en un rango de frecuencia determinado. Este modulo se comercializa en diferentes modelos para trabajar en diferentes bandas ISM (a,b,c,d). se debera tener muy en cuenta la normativa del pais especifico en donde se trabaje, pues no todos los paises permiten utilizar las mismas bandas ISM. En este caso se usara la version de 915 Mhz.

Lora 1276 y Arduino

La comunicacion con el modulo se hace por medio de SPI, en este caso se utilizo un (Clon!) Arduino nano para su configuracion. Ademas de los pines tradicionales del SPI, este modulo requiere el manejo de otras señales adicionales que seran cableadas a pines del arduino. Puesto que el modulo trabaja a 3.3V y el arduino nano trabaja a 5V, se requiere de algun tipo de conversion de nivel entre el arduino y el SX1276. Si no se tiene a la mano dicho integrado se pueden hacer unos divisores de voltaje y trabajar el reloj del SPI en baja velocidad!.

El modulo puede funcionar como transmisor y como receptor, pero no simultaneamente. El codigo es basado en el ejemplo provisto por el fabricante. Este modulo tiene varios multiples configuraciones posibles:



Tipo de modulacion: OOK, FSK y LoRa
Correccion y deteccion de errores: FEC y Cheksum
Consumo de energia: Modos de baja potencia para operacion por baterias y modos potencia total.
Supresion de interferencia: Modos de ensanchamiento de espectro y diferentes factores de frecuencia modulada pulsada

En el ejemplo presentado se utilizo la mayor potencia posible, y los parametros de configuracion que dan la mejor sensibilidad. El ejemplo aqui presentado funciona de la siguiente forma:

El transmisor envia un mensaje cada determinado intervalo de tiempo. En cada envio se conmutara el estado de un led, para poder visualizar cuando se transmite.

El receptor esta atento al mensaje. si este es recibido sin errores, se conmutara el estado de un led. Si se recibe el mensaje, pero tiene algun error, se conmutara el estado de otro led.

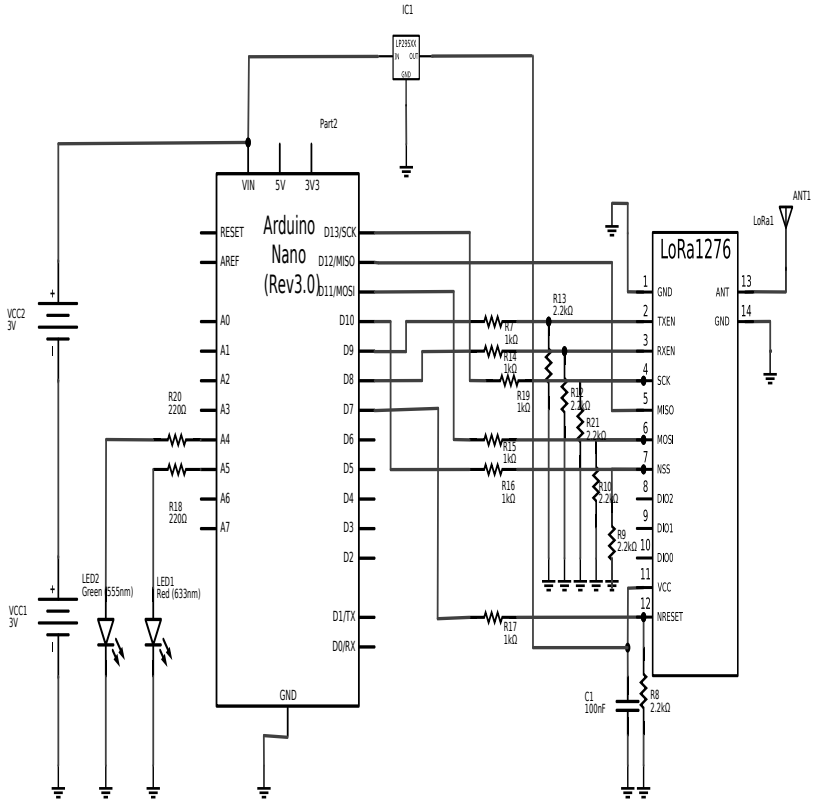
De esta forma es muy facil probar el alcance de los dispositivos. se deja el transmisor en un sitio fijo y se va desplazando el receptor hasta cuando dejen de recibirse mensajes en buen estado o cuando no llegue absolutamente ningun mensaje en el intervalo de tiempo que deberia hacerse.

Este software es una pequeña muestra, pero es la base para construir un sistema mas robusto

CONCLUSIONES

- En el ejemplo se utilizo como antena un alambre cortado a 1/4 de longitud de onda de la señal de transmision (915 Mhz). Con una antena de mejores prestaciones y una ubicacion alta, como en una torre, o en la azotea de un edificio alto, es probable tener coberturas mayores dentro de la ciudad
- Puesto que los modulos solo proporcionan las capas inferiores del modelo OSI, se esta en la libertad de escoger una pila de protocolos de comunicacion ya existente o desarrollar uno propio amoldado a las necesidades
- Para probar el maximo de sensibilidad del receptor (y por lo tanto la maxima distancia de transmision) se requiere de un Oscilador de Cristal Compensado en Temperatura (TCXO) , pues el modulo incorpora un cristal de cuarzo simple
- El circuito aqui presentado no comunica un dispositivo propiamente con internet, sin embargo agregar un modulo Ethernet Arduino al receptor es relativamente simple.

DIAGRAMA ESQUEMATICO



PROGRAMA ARDUINO TRANSMISOR

LORA1276TRANSMISOR.ino

```
/*
Modulo NiceRF LoRa1276 Arduino NANO Clon V3

NANO       LoRa1276
D11 MOSI    6  MOSI
D12 MISO    5  MISO
D13 SCK     4  SCK
D10         7  NSS

por automatizanos.com

*/

// usando la libreria SPI:
#include <SPI.h>

// Definicion de pines digitales
#define MOSI   11
#define MISO   12
#define SCK    13
#define SS     10

#define NRESET 7
#define TXEN   9
#define RXEN   8
#define LED1   A4
#define LED2   A5

// definicion de registros

#define LR_RegFifo                  0x00
#define LR_RegOpMode                0x01
#define LR_RegBitrateMsb            0x02
#define LR_RegBitrateLsb            0x03
#define LR_RegFdevMsb               0x04
#define LR_RegFdMsb                 0x05
#define LR_RegFrMsb                 0x06
#define LR_RegFrMid                 0x07
#define LR_RegFrLsb                 0x08
#define LR_RegPaConfig              0x09
#define LR_RegPaRamp                0x0A
#define LR_RegOcp                   0x0B
#define LR_RegLna                   0x0C
#define LR_RegFifoAddrPtr           0x0D
#define LR_RegFifoTxBaseAddr        0x0E
#define LR_RegFifoRxBaseAddr        0x0F
#define LR_RegFifoRxCurrentaddr     0x10
#define LR_RegIrqFlagsMask          0x11
#define LR_RegIrqFlags              0x12
#define LR_RegRxNbBytes             0x13
#define LR_RegRxHeaderCntValueMsb   0x14
#define LR_RegRxHeaderCntValueLsb   0x15
#define LR_RegRxPacketCntValueMsb   0x16
#define LR_RegRxPacketCntValueLsb   0x17
#define LR_RegModemStat             0x18
#define LR_RegPktSnrValue           0x19
#define LR_RegPktRssiValue          0x1A
#define LR_RegRssiValue             0x1B
#define LR_RegHopChannel            0x1C
#define LR_RegModemConfig1          0x1D
#define LR_RegModemConfig2          0x1E
#define LR_RegSymbTimeoutLsb        0x1F
```

```
#define LR_RegPreambleMsb               0x20
#define LR_RegPreambleLsb               0x21
#define LR_RegPayloadLength             0x22
#define LR_RegMaxPayloadLength          0x23
#define LR_RegHopPeriod                 0x24
#define LR_RegFifoRxByteAddr            0x25
#define LR_RegModemConfig3              0x26
#define REG_LR_DIOMAPPING1              0x40
#define REG_LR_DIOMAPPING2              0x41
#define REG_LR_VERSION                  0x42
#define REG_LR_PLLHOP                   0x44
#define REG_LR_TCXO                     0x4B
#define REG_LR_PADAC                    0x4D
#define REG_LR_FORMERTEMP               0x5B
#define REG_LR_AGCREF                   0x61
#define REG_LR_AGCTHRESH1               0x62
#define REG_LR_AGCTHRESH2               0x63
#define REG_LR_AGCTHRESH3               0x64

// longitud del area de datos
#define longitud_areadatos  6

// paquete transmision
unsigned char txbuf[longitud_areadatos]={'p','r','u','e','b','a'};
// paquete recepcion
unsigned char rxbuf[30];


// Inicializaciones
void setup() {
  byte temporal = 0;


  // Iniciando el puerto serial para depuracion en caso de necesitarse
  Serial.begin(9600);


  // Inicializando los pines del SPI como corresponde

  pinMode(MOSI, OUTPUT);
  pinMode(MISO, INPUT);
  pinMode(SCK,OUTPUT);
  pinMode(SS,OUTPUT);
  digitalWrite(SS,HIGH); //deshabilitando el modulo LoRa

  // Inicializando otros pines del modulo
  pinMode(NRESET, OUTPUT);
  pinMode(TXEN, OUTPUT);
  pinMode(RXEN, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);

  digitalWrite(NRESET,HIGH); // Sacar de reset el modulo
  digitalWrite(TXEN,LOW); // Desconectando circuito de antena transmision
  digitalWrite(RXEN,LOW); // Desconectando circuito de antena recepcion
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);

  /* Inicializando los registros internos de manejo del SPI
  descripcion de cada uno de los bits del registro SPCR
  | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
  | SPIE | SPE  | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |

  SPIE - Habilita la interrupcion del SPI cuando vale 1
  SPE  - Habilita el SPI cuando vale 1
  DORD - Envia el bit MENOS significativo (LSB) de pimero cuando vale 1, envia el MAS significativo (MSB) cuando vale 0
  MSTR - Configura el arduino en modo SPI maestro cuando vale 1, en modo esclavo cuando vale 0
  CPOL - Configura la señaldel reloj para inactiva en nivel alto cuando vale 1, inactiva en nivel bajo si vale 0
  CPHA - Lee los datos en el flanco descendiente del reloj cuando vale 1, en el flanco de subidca cuando vale 0
  SPR1 y SPR0 - Configuran la velocidad del SPI, 00 la mas rapida (4MHz) 11 la mas lenta (250KHz)
```

```
  // SPCR = 01010011
  //interrupcion deshabilitada,spi habilitado,bit mas significatibo (msb) de primero,SPI maestro,reloj inactivo en bajo,
  toma de datos en el flanco de subida del reloj,velocidad del reloj la mas lenta*/

  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1)|(1<<SPR0);
  temporal=SPSR;  //Leyendo y descartando posible basura de ejecuciones anteriores
  temporal=SPDR;  //Leyendo y descartando posible basura de ejecuciones anteriores
  delay(10);


}

void loop() {

  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);

  reset_sx1276();
  Configuracion_SX1276();  // inicializando el modulo RF


  while(1){
    modo_transmision();                 // transmitir paquete
    delay(300);

  }

}



byte SPIleeRegistro(byte direccion) {

  byte resultado;

  digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

  SPDR = direccion;               // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 0! para sx1276
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Desechar la primera lectura

  SPDR = 0x0;                     // Enviando cualquier dato por el bus para traer el resultado
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Obteniendo el valor del registro

  digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

  return (resultado);

}

byte SPIescribeRegistro(byte direccion,byte valor) {

  byte resultado;

  digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

  SPDR = direccion | 0x80;        // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para sx1276
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Desechar la primera lectura

  SPDR = valor;                   // Enviando cualquier dato por el bus para traer el resultado
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
```

```
  resultado = SPDR;               // Desechar la segunda lectura

  digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

}

void SPIescribeRafaga(unsigned char direccion, unsigned char *ptr, unsigned char longitud)
{
                unsigned char i;
                unsigned char resultado;

        digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

        SPDR = direccion | 0x80;        // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para
sx1276
        while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
        {
        };
        resultado = SPDR;               // Desechar la primera lectura

        for (i=0; i <= longitud; i++){

          SPDR = *ptr;                  // Enviando cualquier dato por el bus para traer el resultado
          while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
          {
          };
          resultado = SPDR;             // Desechar la segunda lectura

          //DEBUG DEBUG DEBUG
          Serial.print(*ptr, HEX);
          //DEBUG DEBUG DEBUG

          ptr++;
        }

        //DEBUG DEBUG DEBUG
        Serial.print("\n");
        // DEBUG DEBUG DEBUG

        digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

}
void SPIleeRafaga(unsigned char direccion, unsigned char *ptr, unsigned char longitud)
{
                unsigned char i;
                unsigned char resultado;

        digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

        SPDR = direccion;               // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para
sx1276
        while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
        {
        };
        resultado = SPDR;               // Desechar la primera lectura

        for (i=0; i <= longitud; i++){

          SPDR = 0;                     // Enviando cualquier dato por el bus para traer el resultado
          while (!(SPSR & (1<<SPIF)))   // Espera por el fin de la transmision
          {
          };
          *ptr = SPDR;                  // Escribir en el apuntador


          ptr++;
        }

        //DEBUG DEBUG DEBUG
        Serial.print("\n");
        // DEBUG DEBUG DEBUG
```

```
    digitalWrite(SS, HIGH);       // Deseleccionar el modulo LoRa

}

void reset_sx1276(void)
{
  digitalWrite(TXEN, LOW);
  digitalWrite(RXEN, LOW);

  digitalWrite(NRESET, LOW);
  delay(10);
  digitalWrite(NRESET, HIGH);
  delay(20);
}

void Configuracion_SX1276(void)
{
        // Para configurar el modulo se debe poner en modo sleep

        SPIescribeRegistro(LR_RegOpMode,0x00);          // modo sleep, alta frecuencia
        delay(10);

        SPIescribeRegistro(REG_LR_TCXO,0x09);           // Cristal externo
    SPIescribeRegistro(LR_RegOpMode,0x80);              // modo LoRa, alta frecuencia

    SPIescribeRegistro(LR_RegFrMsb,0xE4);
        SPIescribeRegistro(LR_RegFrMid,0xC0);
        SPIescribeRegistro(LR_RegFrLsb,0x00);           // frequency：915hz


        SPIescribeRegistro(LR_RegPaConfig,0xFF);        // maxima potencia de salida PA_BOOST habilitado


        SPIescribeRegistro(LR_RegOcp,0x0B);             // cerrar protecion de sobre corriente  ocp
        SPIescribeRegistro(LR_RegLna,0x23);             // habilitar LNA

    SPIescribeRegistro(LR_RegModemConfig1,0x72);        // ancho de banda de señal：125kHz,codificacion de error = 4/5, modo
cabecera explicita

        SPIescribeRegistro(LR_RegModemConfig2,0xC7);    // factor de ensanchamiento：12

        SPIescribeRegistro(LR_RegModemConfig3,0x08);    // LNA? optimizado para velocidad de transmision lenta

        SPIescribeRegistro(LR_RegSymbTimeoutLsb,0xFF);  // maximo tiempo de espera recepcion

        SPIescribeRegistro(LR_RegPreambleMsb,0x00);
        SPIescribeRegistro(LR_RegPreambleLsb,16);       // preambulo 16 bytes

        SPIescribeRegistro(REG_LR_PADAC,0x87);          // potencia de transmision 20dBm
        SPIescribeRegistro(LR_RegHopPeriod,0x00);       // sin saltos de frecuencia

        SPIescribeRegistro(REG_LR_DIOMAPPING2,0x01);    // DIO5=ModeReady,DIO4=CadDetected
        SPIescribeRegistro(LR_RegOpMode,0x01);          // modo standby, alta frecuencia
}

void modo_transmision(void)
{
        unsigned char direccion,temporal;

        digitalWrite(TXEN,HIGH);                // abrir conmutador de antena transmision
    digitalWrite(RXEN,LOW);


        SPIescribeRegistro(REG_LR_DIOMAPPING1,0x41);            // DIO0=TxDone,DIO1=RxTimeout,DIO3=ValidHeader

        SPIescribeRegistro(LR_RegIrqFlags,0xff);        // limpiando interrupcion
        SPIescribeRegistro(LR_RegIrqFlagsMask,0xf7);            // habilitando txdone
        SPIescribeRegistro(LR_RegPayloadLength,longitud_areadatos);  // longitud de paquete de datos

        direccion = SPIleeRegistro(LR_RegFifoTxBaseAddr);  // read TxBaseAddr
```

```
        SPIescribeRegistro(LR_RegFifoAddrPtr,direccion);      // TxBaseAddr->FifoAddrPtr

        SPIescribeRafaga(0x00,txbuf,longitud_areadatos);      // escribir los datos en la fifo
        SPIescribeRegistro(LR_RegOpMode,0x03);                // modo transmision, alta frequency


    digitalWrite(LED1, !digitalRead(LED1));


        temporal=SPIleeRegistro(LR_RegIrqFlags);              // leer la interrupcion
        while(!(temporal&0x08))                               // esperar por bandera txdone
        {
                temporal=SPIleeRegistro(LR_RegIrqFlags);

        }

        digitalWrite(TXEN,LOW);                 // cerrar conmutador de antena transmision
    digitalWrite(RXEN,LOW);

        SPIescribeRegistro(LR_RegIrqFlags,0xff);              // limpiar interrupcion
        SPIescribeRegistro(LR_RegOpMode,0x01);                // modo standby, alta frecuencia

}

void inicializa_recepcion(void)
{
        unsigned char direccion;

    digitalWrite(TXEN,LOW);                 // abrir el conmutador de recepcion de antena
    digitalWrite(RXEN,HIGH);

        SPIescribeRegistro(REG_LR_DIOMAPPING1,0x01);          //DIO0=00, DIO1=00, DIO2=00, DIO3=01  DIO0=00--RXDONE

        SPIescribeRegistro(LR_RegIrqFlagsMask,0x3f);          // habilitar rxdone y rxtimeout
        SPIescribeRegistro(LR_RegIrqFlags,0xff);              // limpiar la interrupcion

        direccion = SPIleeRegistro(LR_RegFifoRxBaseAddr);  // read RxBaseAddr
        SPIescribeRegistro(LR_RegFifoAddrPtr,direccion);      // RxBaseAddr->FifoAddrPtr
        SPIescribeRegistro(LR_RegOpMode,0x05);                // modo recepcion continuo alta frecuencia
}
```

PROGRAMA ARDUINO RECEPTOR

LORA1276RECEPTOR.ino

```
/*
Modulo NiceRF LoRa1276 Arduino NANO Clon V3

NANO       LoRa1276
D11 MOSI    6  MOSI
D12 MISO    5  MISO
D13 SCK     4  SCK
D10         7  NSS

por automatizanos.com

*/

// usando la libreria SPI:
#include <SPI.h>

// Definicion de pines digitales
#define MOSI   11
#define MISO   12
#define SCK    13
#define SS     10

#define NRESET 7
#define TXEN   9
#define RXEN   8
#define LED1   A4
#define LED2   A5

// definicion de registros

#define LR_RegFifo                  0x00
#define LR_RegOpMode                0x01
#define LR_RegBitrateMsb            0x02
#define LR_RegBitrateLsb            0x03
#define LR_RegFdevMsb               0x04
#define LR_RegFdMsb                 0x05
#define LR_RegFrMsb                 0x06
#define LR_RegFrMid                 0x07
#define LR_RegFrLsb                 0x08
#define LR_RegPaConfig              0x09
#define LR_RegPaRamp                0x0A
#define LR_RegOcp                   0x0B
#define LR_RegLna                   0x0C
#define LR_RegFifoAddrPtr           0x0D
#define LR_RegFifoTxBaseAddr        0x0E
#define LR_RegFifoRxBaseAddr        0x0F
#define LR_RegFifoRxCurrentaddr     0x10
#define LR_RegIrqFlagsMask          0x11
#define LR_RegIrqFlags              0x12
#define LR_RegRxNbBytes             0x13
#define LR_RegRxHeaderCntValueMsb   0x14
#define LR_RegRxHeaderCntValueLsb   0x15
#define LR_RegRxPacketCntValueMsb   0x16
#define LR_RegRxPacketCntValueLsb   0x17
#define LR_RegModemStat             0x18
#define LR_RegPktSnrValue           0x19
#define LR_RegPktRssiValue          0x1A
#define LR_RegRssiValue             0x1B
#define LR_RegHopChannel            0x1C
#define LR_RegModemConfig1          0x1D
#define LR_RegModemConfig2          0x1E
#define LR_RegSymbTimeoutLsb        0x1F
```

```
#define LR_RegPreambleMsb               0x20
#define LR_RegPreambleLsb               0x21
#define LR_RegPayloadLength             0x22
#define LR_RegMaxPayloadLength          0x23
#define LR_RegHopPeriod                 0x24
#define LR_RegFifoRxByteAddr            0x25
#define LR_RegModemConfig3              0x26
#define REG_LR_DIOMAPPING1              0x40
#define REG_LR_DIOMAPPING2              0x41
#define REG_LR_VERSION                  0x42
#define REG_LR_PLLHOP                   0x44
#define REG_LR_TCXO                     0x4B
#define REG_LR_PADAC                    0x4D
#define REG_LR_FORMERTEMP               0x5B
#define REG_LR_AGCREF                   0x61
#define REG_LR_AGCTHRESH1               0x62
#define REG_LR_AGCTHRESH2               0x63
#define REG_LR_AGCTHRESH3               0x64

// longitud del area de datos
#define longitud_areadatos  6

// paquete transmision
unsigned char txbuf[longitud_areadatos]={'p','r','u','e','b','a'};
// paquete recepcion
unsigned char rxbuf[30];


// Inicializaciones
void setup() {
  byte temporal = 0;


  // Iniciando el puerto serial para depuracion en caso de necesitarse
  Serial.begin(9600);


  // Inicializando los pines del SPI como corresponde

  pinMode(MOSI, OUTPUT);
  pinMode(MISO, INPUT);
  pinMode(SCK,OUTPUT);
  pinMode(SS,OUTPUT);
  digitalWrite(SS,HIGH); //deshabilitando el modulo LoRa

  // Inicializando otros pines del modulo
  pinMode(NRESET, OUTPUT);
  pinMode(TXEN, OUTPUT);
  pinMode(RXEN, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);

  digitalWrite(NRESET,HIGH); // Sacar de reset el modulo
  digitalWrite(TXEN,LOW); // Desconectando circuito de antena transmision
  digitalWrite(RXEN,LOW); // Desconectando circuito de antena recepcion
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);

  /* Inicializando los registros internos de manejo del SPI
  descripcion de cada uno de los bits del registro SPCR
  | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
  | SPIE | SPE  | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |

  SPIE - Habilita la interrupcion del SPI cuando vale 1
  SPE  - Habilita el SPI cuando vale 1
  DORD - Envia el bit MENOS significativo (LSB) de pimero cuando vale 1, envia el MAS significativo (MSB) cuando vale 0
  MSTR - Configura el arduino en modo SPI maestro cuando vale 1, en modo esclavo cuando vale 0
  CPOL - Configura la señaldel reloj para inactiva en nivel alto cuando vale 1, inactiva en nivel bajo si vale 0
  CPHA - Lee los datos en el flanco descendiente del reloj cuando vale 1, en el flanco de subidca cuando vale 0
  SPR1 y SPR0 - Configuran la velocidad del SPI, 00 la mas rapida (4MHz) 11 la mas lenta (250KHz)
```

```
  // SPCR = 01010011
  //interrupcion deshabilitada,spi habilitado,bit mas significatibo (msb) de primero,SPI maestro,reloj inactivo en bajo,
  toma de datos en el flanco de subida del reloj,velocidad del reloj la mas lenta*/

  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1)|(1<<SPR0);
  temporal=SPSR;  //Leyendo y descartando posible basura de ejecuciones anteriores
  temporal=SPDR;  //Leyendo y descartando posible basura de ejecuciones anteriores
  delay(10);


}

void loop() {

  unsigned char temporal,longitud_paquete;

  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);

  reset_sx1276();
  Configuracion_SX1276();                       // initializar el modulo RF

  inicializa_recepcion();                     // modo transmision

  while(1){
    temporal=SPIleeRegistro(LR_RegIrqFlags);                  // leer la interrupcion
    if(temporal & 0x40){                      // esperar la bandera rxdone
      SPIescribeRegistro(LR_RegIrqFlags,0xff);                        // limpiar interrupcion
      temporal = SPIleeRegistro(LR_RegFifoRxCurrentaddr);          // read RxCurrentaddr
      SPIescribeRegistro(LR_RegFifoAddrPtr,temporal);               // RxCurrentaddr -> FiFoAddrPtr

      longitud_paquete = SPIleeRegistro(LR_RegRxNbBytes);         // leer la longitud del paquete
      SPIleeRafaga(0x00, rxbuf, longitud_paquete);              // leer desde la fifo

      //"prueba"
      if( (rxbuf[0] == 'p') && (rxbuf[5] == 'a') )                  // verificacion simple del paquete, usar el registro de CRC para mayor robustez
      {
              digitalWrite(LED2, !digitalRead(LED2));                       // si los datos son correctos conmutar LED2
              inicializa_recepcion();
      }
      else
      {
        digitalWrite(LED1, !digitalRead(LED1));                     // si los datos NO son correctos conmutar LED1
        inicializa_recepcion();                                     // reiniciar el receptor cuando hay falla
      }
    }

  }


}



byte SPIleeRegistro(byte direccion) {

  byte resultado;

  digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

  SPDR = direccion;               // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 0! para sx1276
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Desechar la primera lectura

  SPDR = 0x0;                   // Enviando cualquier dato por el bus para traer el resultado
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Obteniendo el valor del registro
```

```
  digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

  return (resultado);

}

byte SPIescribeRegistro(byte direccion,byte valor) {

  byte resultado;

  digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

  SPDR = direccion | 0x80;        // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para sx1276
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Desechar la primera lectura

  SPDR = valor;                   // Enviando cualquier dato por el bus para traer el resultado
  while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
  {
  };
  resultado = SPDR;               // Desechar la segunda lectura

  digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

}

void SPIescribeRafaga(unsigned char direccion, unsigned char *ptr, unsigned char longitud)
{
        unsigned char i;
        unsigned char resultado;

    digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa

    SPDR = direccion | 0x80;        // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para
sx1276
    while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
    {
    };
    resultado = SPDR;               // Desechar la primera lectura

    for (i=0; i <= longitud; i++){

      SPDR = *ptr;                  // Enviando cualquier dato por el bus para traer el resultado
      while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
      {
      };
      resultado = SPDR;             // Desechar la segunda lectura

      //DEBUG DEBUG DEBUG
      Serial.print(*ptr, HEX);
      //DEBUG DEBUG DEBUG

      ptr++;
    }

    //DEBUG DEBUG DEBUG
    Serial.print("\n");
    // DEBUG DEBUG DEBUG

    digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

}
void SPIleeRafaga(unsigned char direccion, unsigned char *ptr, unsigned char longitud)
{
        unsigned char i;
        unsigned char resultado;

    digitalWrite(SS, LOW);          // Seleccionar el modulo LoRa
```

```
      SPDR = direccion;                // Envia la direccion & Inicia Transmision. En modo LECTURA el bit 7 de la direccion siempre es 1! para
sx1276
      while (!(SPSR & (1<<SPIF)))     // Espera por el fin de la transmision
      {
      };
      resultado = SPDR;               // Desechar la primera lectura

      for (i=0; i <= longitud; i++){

        SPDR = 0;                     // Enviando cualquier dato por el bus para traer el resultado
        while (!(SPSR & (1<<SPIF)))   // Espera por el fin de la transmision
        {
        };
        *ptr = SPDR;                  // Escribir en el apuntador


        ptr++;
      }

      //DEBUG DEBUG DEBUG
      Serial.print("\n");
      // DEBUG DEBUG DEBUG

      digitalWrite(SS, HIGH);         // Deseleccionar el modulo LoRa

}

void reset_sx1276(void)
{
   digitalWrite(TXEN, LOW);
   digitalWrite(RXEN, LOW);

   digitalWrite(NRESET, LOW);
   delay(10);
   digitalWrite(NRESET, HIGH);
   delay(20);
}

void Configuracion_SX1276(void)
{
            // Para configurar el modulo se debe poner en modo sleep

            SPIescribeRegistro(LR_RegOpMode,0x00);              // modo sleep, alta frecuencia
            delay(10);

            SPIescribeRegistro(REG_LR_TCXO,0x09);               // Cristal externo
      SPIescribeRegistro(LR_RegOpMode,0x80);                    // modo LoRa, alta frecuencia

      SPIescribeRegistro(LR_RegFrMsb,0xE4);
            SPIescribeRegistro(LR_RegFrMid,0xC0);
            SPIescribeRegistro(LR_RegFrLsb,0x00);               // frequency：915hz


            SPIescribeRegistro(LR_RegPaConfig,0xFF);            // maxima potencia de salida PA_BOOST habilitado


            SPIescribeRegistro(LR_RegOcp,0x0B);                 // cerrar protecion de sobre corriente  ocp
            SPIescribeRegistro(LR_RegLna,0x23);                 // habilitar LNA

      SPIescribeRegistro(LR_RegModemConfig1,0x72);              // ancho de banda de señal：125kHz,codificacion de error = 4/5, modo
cabecera explicita

            SPIescribeRegistro(LR_RegModemConfig2,0xC7);      // factor de ensanchamiento：12

            SPIescribeRegistro(LR_RegModemConfig3,0x08);      // LNA? optimizado para velocidad de transmision lenta

            SPIescribeRegistro(LR_RegSymbTimeoutLsb,0xFF);    // maximo tiempo de espera recepcion

            SPIescribeRegistro(LR_RegPreambleMsb,0x00);
            SPIescribeRegistro(LR_RegPreambleLsb,16);         // preambulo 16 bytes
```

```
	SPIescribeRegistro(REG_LR_PADAC,0x87);		// potencia de transmision 20dBm
	SPIescribeRegistro(LR_RegHopPeriod,0x00);	// sin saltos de frecuencia

	SPIescribeRegistro(REG_LR_DIOMAPPING2,0x01);	// DIO5=ModeReady,DIO4=CadDetected
	SPIescribeRegistro(LR_RegOpMode,0x01);		// modo standby, alta frecuencia
}

void modo_transmision(void)
{
	unsigned char direccion,temporal;

	digitalWrite(TXEN,HIGH);			// abrir conmutador de antena transmision
    digitalWrite(RXEN,LOW);


	SPIescribeRegistro(REG_LR_DIOMAPPING1,0x41);		// DIO0=TxDone,DIO1=RxTimeout,DIO3=ValidHeader

	SPIescribeRegistro(LR_RegIrqFlags,0xff);		// limpiando interrupcion
	SPIescribeRegistro(LR_RegIrqFlagsMask,0xf7);		// habilitando txdone
	SPIescribeRegistro(LR_RegPayloadLength,longitud_areadatos);	// longitud de paquete de datos

	direccion = SPIleeRegistro(LR_RegFifoTxBaseAddr);	// read TxBaseAddr
	SPIescribeRegistro(LR_RegFifoAddrPtr,direccion);	// TxBaseAddr->FifoAddrPtr

	SPIescribeRafaga(0x00,txbuf,longitud_areadatos);	// escribir los datos en la fifo
	SPIescribeRegistro(LR_RegOpMode,0x03);			// modo transmision, alta frequency


    digitalWrite(LED1, !digitalRead(LED1));


	temporal=SPIleeRegistro(LR_RegIrqFlags);		// leer la interrupcion
	while(!(temporal&0x08))					// esperar por bandera txdone
	{
		temporal=SPIleeRegistro(LR_RegIrqFlags);

	}

	digitalWrite(TXEN,LOW);				// cerrar conmutador de antena transmision
    digitalWrite(RXEN,LOW);

	SPIescribeRegistro(LR_RegIrqFlags,0xff);		// limpiar interrupcion
	SPIescribeRegistro(LR_RegOpMode,0x01);			// modo standby, alta frecuencia

}

void inicializa_recepcion(void)
{
	unsigned char direccion;

    digitalWrite(TXEN,LOW);				// abrir el conmutador de recepcion de antena
    digitalWrite(RXEN,HIGH);

	SPIescribeRegistro(REG_LR_DIOMAPPING1,0x01);		//DIO0=00, DIO1=00, DIO2=00, DIO3=01  DIO0=00--RXDONE

	SPIescribeRegistro(LR_RegIrqFlagsMask,0x3f);		// habilitar rxdone y rxtimeout
	SPIescribeRegistro(LR_RegIrqFlags,0xff);		// limpiar la interrupcion

	direccion = SPIleeRegistro(LR_RegFifoRxBaseAddr);	// read RxBaseAddr
	SPIescribeRegistro(LR_RegFifoAddrPtr,direccion);	// RxBaseAddr->FifoAddrPtr
	SPIescribeRegistro(LR_RegOpMode,0x05);			// modo recepcion continuo alta frecuencia
}
```