

---

```
/******
```

```
Title:sx1276_7_8 demo code
```

```
Current version:v1.0
```

```
Function:demo
```

```
Processor Arduino
```

```
Clock:
```

```
Operate frequency:433MHZ band\
```

```
Date rate:for programme
```

```
modulation: Lora/
```

```
deviation:for programme
```

```
Transmit one packet data time : Indefinite package
```

```
Work mode:
```

```
company:WWW.DORJI.COM
```

```
author:DORJI
```

```
Contact:
```

```
Date:2014-07-02
```

```
*****/
```

```
unsigned char mode;//lora--1/FSK--0
```

```
unsigned char Freq_Sel;//
```

```
unsigned char Power_Sel;//
```

```
unsigned char Lora_Rate_Sel;//
```

```
unsigned char BandWide_Sel;//
```

```
unsigned char Fsk_Rate_Sel;//
```

```
int led = 13;
```

```
int nsel = 22;
```

```
int sck = 24;
```

```
int mosi = 26;
```

```
int miso = 28;
```

```
int dio0 = 30;
```

```
int reset = 32;
```

```
/******
```

```
**Name: SPICmd8bit
```

```
**Function: SPI Write one byte
```

```
**Input: WrPara
```

```
**Output: none
```

```
**note: use for burst mode
```

```
*****/
```

```
void SPICmd8bit(unsigned char WrPara)
```

```
{
```

```
    unsigned char bitcnt;
```

```
    digitalWrite(nsel, LOW);//nSEL_L();
```

---

```

digitalWrite(sck, LOW); //SCK_L();

for(bitcnt=8; bitcnt!=0; bitcnt--)
{
    digitalWrite(sck, LOW); //SCK_L();

    if(WrPara&0x80)
        digitalWrite(mosi, HIGH); //SDI_H();
    else
        digitalWrite(mosi, LOW); //SDI_L();

    digitalWrite(sck, HIGH); //SCK_H();

    WrPara <<= 1;
}
digitalWrite(sck, LOW); //SCK_L();
digitalWrite(mosi, HIGH); //SDI_H();
}

/*****
**Name:      SPIRead8bit
**Function:  SPI Read one byte
**Input:     None
**Output:    result byte
**Note:      use for burst mode
*****/
unsigned char SPIRead8bit(void)
{
    unsigned char RdPara = 0;
    unsigned char bitcnt;

    digitalWrite(nsel, LOW); //nSEL_L();
    digitalWrite(mosi, HIGH); //SDI_H(); //Read one byte data from FIFO, MOSI hold
to High
    for(bitcnt=8; bitcnt!=0; bitcnt--)
    {
        digitalWrite(sck, LOW); //SCK_L();
        RdPara <<= 1;
        digitalWrite(sck, HIGH); //SCK_H();

        if(digitalRead(miso)) //if(Get_SD0())
            RdPara |= 0x01;
        else
            RdPara |= 0x00;
    }
}

```

---

```

    }
    digitalWrite(sck, LOW); //SCK_L();
    return(RdPara);
}

/*****
**Name:      SPIRead
**Function:  SPI Read CMD
**Input:     adr -> address for read
**Output:    None
*****/
unsigned char SPIRead(unsigned char adr)
{
    unsigned char tmp;

    SPICmd8bit(adr); //Send address first
    tmp = SPIRead8bit();
    digitalWrite(nsel, HIGH); //nSEL_H();
    return(tmp);
}

/*****
**Name:      SPIWrite
**Function:  SPI Write CMD
**Input:     u8 address & u8 data
**Output:    None
*****/
void SPIWrite(unsigned char adr, unsigned char WrPara)
{
    digitalWrite(nsel, LOW); //nSEL_L();

    SPICmd8bit(adr|0x80); //写入地址
    SPICmd8bit(WrPara); //写入数据

    digitalWrite(sck, LOW); //SCK_L();
    digitalWrite(mosi, HIGH); //SDI_H();
    digitalWrite(nsel, HIGH); //nSEL_H();
}

/*****
**Name:      SPIBurstRead
**Function:  SPI burst read mode
**Input:     adr-----address for read
**           ptr-----data buffer point for read
**           length--how many bytes for read
*****/

```

---

```

**Output:   None
*****/
void SPIBurstRead(unsigned char adr, unsigned char *ptr, unsigned char leng)
{
    unsigned char i;
    if(leng<=1)                                     //length must more than
one
        return;
    else
    {
        digitalWrite(sck, LOW); //SCK_L();
        digitalWrite(nsel, LOW); //nSEL_L();

        SPICmd8bit(adr);
        for(i=0;i<leng;i++)
            ptr[i] = SPIRead8bit();
        digitalWrite(nsel, HIGH); //nSEL_H();
    }
}

/*****
**Name:      SPIBurstWrite
**Function:  SPI burst write mode
**Input:     adr-----address for write
**           ptr-----data buffer point for write
**           length--how many bytes for write
**Output:    none
*****/
void BurstWrite(unsigned char adr, unsigned char *ptr, unsigned char leng)
{
    unsigned char i;

    if(leng<=1)                                     //length must more than
one
        return;
    else
    {
        digitalWrite(sck, LOW); //SCK_L();
        digitalWrite(nsel, LOW); //nSEL_L();
        SPICmd8bit(adr|0x80);
        for(i=0;i<leng;i++)
            SPICmd8bit(ptr[i]);
        digitalWrite(nsel, HIGH); //nSEL_H();
    }
}

```

---

```

}
/*****
// RF module:          sx1276_7_8
// FSK:
// Carry Frequency:    434MHz
// Bit Rate:           1.2Kbps/2.4Kbps/4.8Kbps/9.6Kbps
// Tx Power Output:    20dbm/17dbm/14dbm/11dbm
// Frequency Deviation: +/-35KHz
// Receive Bandwidth:  83KHz
// Coding:             NRZ
// Packet Format:       0x5555555555+0xAA2DD4+"Mark1 Lora sx1276_7_8" (total: 29
bytes)
// LoRa:
// Carry Frequency:    434MHz
// Spreading Factor:   6/7/8/9/10/11/12
// Tx Power Output:    20dbm/17dbm/14dbm/11dbm
//                                     Receive          Bandwidth:
7.8KHz/10.4KHz/15.6KHz/20.8KHz/31.2KHz/41.7KHz/62.5KHz/125KHz/250KHz/500KHz
// Coding:             NRZ
// Packet Format:       "HR_WT Lora sx1276_7_8" (total: 21 bytes)
// Tx Current:         about 120mA (RFOP=+20dBm, typ.)
// Rx Current:         about 11.5mA (typ.)
*****/
////////////////////////////////////// LoRa mode
//////////////////////////////////////
//Error Coding rate (CR)setting
#define CR_4_5
//#define CR_4_6    0
//#define CR_4_7    0
//#define CR_4_8    0

#ifdef CR_4_5
#define CR    0x01 // 4/5
#else
#ifdef CR_4_6
#define CR    0x02 // 4/6
#else
#ifdef CR_4_7
#define CR    0x03 // 4/7
#else
#ifdef CR_4_8
#define CR    0x04 // 4/8
#endif
#endif
#endif
#endif

```

---

```

    #endif
#endif

//CRC Enable
#define CRC_EN

#ifdef CRC_EN
    #define CRC    0x01                                //CRC Enable
#else
    #define CRC    0x00
#endif

//RFM98 Internal registers Address
/*****Lroa mode*****/
#define LR_RegFifo                                0x00
// Common settings
#define LR_RegOpMode                                0x01
#define LR_RegFrMsb                                0x06
#define LR_RegFrMid                                0x07
#define LR_RegFrLsb                                0x08
// Tx settings
#define LR_RegPaConfig                                0x09
#define LR_RegPaRamp                                0x0A
#define LR_RegOcp                                    0x0B
// Rx settings
#define LR_RegLna                                    0x0C
// LoRa registers
#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_RegRxBnBytes                             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

// I/O settings
#define REG_LR_DIOMAPPING1                 0x40
#define REG_LR_DIOMAPPING2                 0x41
// Version
#define REG_LR_VERSION                     0x42
// Additional settings
#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

/*****FSK/ook mode*****/
#define RegFIFO                0x00           //FIFO
#define RegOpMode               0x01           //operation mode
#define RegBitRateMsb           0x02           //BR MSB
#define RegBitRateLsb           0x03           //BR LSB
#define RegFdevMsb              0x04           //FD MSB
#define RegFdevLsb              0x05           //FD LSB
#define RegFreqMsb              0x06           //Freq MSB
#define RegFreqMid              0x07           //Freq Middle
#define RegFreqLsb              0x08           //Freq LSB
#define RegPaConfig             0x09
#define RegPaRamp               0x0a
#define RegOcp                  0x0b
#define RegLna                  0x0c
#define RegRxConfig             0x0d
#define RegRssiConfig           0x0e
#define RegRssiCollision        0x0f
#define RegRssiThresh           0x10
#define RegRssiValue            0x11

```

---

```
#define RegRxBw 0x12
#define RegAfcBw 0x13
#define RegOokPeak 0x14
#define RegOokFix 0x15
#define RegOokAvg 0x16

#define RegAfcFei 0x1a
#define RegAfcMsb 0x1b
#define RegAfcLsb 0x1c
#define RegFeiMsb 0x1d
#define RegFeiLsb 0x1e
#define RegPreambleDetect 0x1f
#define RegRxTimeout1 0x20
#define RegRxTimeout2 0x21
#define RegRxTimeout3 0x22
#define RegRxDelay 0x23
#define RegOsc 0x24 //Set OSC
#define RegPreambleMsb 0x25
#define RegPreambleLsb 0x26
#define RegSyncConfig 0x27
#define RegSyncValue1 0x28
#define RegSyncValue2 0x29
#define RegSyncValue3 0x2a
#define RegSyncValue4 0x2b
#define RegSyncValue5 0x2c
#define RegSyncValue6 0x2d
#define RegSyncValue7 0x2e
#define RegSyncValue8 0x2f
#define RegPacketConfig1 0x30
#define RegPacketConfig2 0x31
#define RegPayloadLength 0x32
#define RegNodeAdrs 0x33
#define RegBroadcastAdrs 0x34
#define RegFifoThresh 0x35
#define RegSeqConfig1 0x36
#define RegSeqConfig2 0x37
#define RegTimerResol 0x38
#define RegTimer1Coef 0x39
#define RegTimer2Coef 0x3a
#define RegImageCal 0x3b
#define RegTemp 0x3c
#define RegLowBat 0x3d
#define RegIrqFlags1 0x3e
#define RegIrqFlags2 0x3f
```



---

```

#define RegDioMapping1      0x40
#define RegDioMapping2      0x41
#define RegVersion           0x42

#define RegPllHop            0x44
#define RegPaDac             0x4d
#define RegBitRateFrac       0x5d
/*****
**Parameter table define
*****/
unsigned char sx1276_7_8FreqTbl[1][3] =
{
    {0x6C, 0x80, 0x00}, //434MHz
};

unsigned char sx1276_7_8PowerTbl[4] =
{
    0xFF,           //20dbm
    0xFC,           //17dbm
    0xF9,           //14dbm
    0xF6,           //11dbm
};

unsigned char sx1276_7_8SpreadFactorTbl[7] =
{
    6, 7, 8, 9, 10, 11, 12
};

unsigned char sx1276_7_8LoRaBwTbl[10] =
{
    //7.8KHz, 10.4KHz, 15.6KHz, 20.8KHz, 31.2KHz, 41.7KHz, 62.5KHz, 125KHz, 250KHz, 500KHz
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9
};

unsigned char sx1276_7_8Data[] = {"Mark1 Lora sx1276_7_8"};

unsigned char RxData[64];

/*****
**Name:      sx1276_7_8_Standby
**Function:  Entry standby mode
**Input:     None
**Output:    None
*****/

```

---

```

void sx1276_7_8_Standby(void)
{
    SPIWrite(LR_RegOpMode, 0x09);
    //Standby//Low Frequency Mode
    //SPIWrite(LR_RegOpMode, 0x01);
//Standby//High Frequency Mode
}

/*****
**Name:      sx1276_7_8_Sleep
**Function:  Entry sleep mode
**Input:     None
**Output:    None
*****/
void sx1276_7_8_Sleep(void)
{
    SPIWrite(LR_RegOpMode, 0x08);                                     //Sleep//Low
Frequency Mode
    //SPIWrite(LR_RegOpMode, 0x00);
//Sleep//High Frequency Mode
}

/*****
//LoRa mode
*****/
/*****
**Name:      sx1276_7_8_EntryLoRa
**Function:  Set RFM69 entry LoRa(LongRange) mode
**Input:     None
**Output:    None
*****/
void sx1276_7_8_EntryLoRa(void)
{
    SPIWrite(LR_RegOpMode, 0x88); //Low Frequency Mode
    //SPIWrite(LR_RegOpMode, 0x80); //High Frequency Mode
}

/*****
**Name:      sx1276_7_8_LoRaClearIrq
**Function:  Clear all irq
**Input:     None
**Output:    None
*****/
void sx1276_7_8_LoRaClearIrq(void)

```

---

```

{
    SPIWrite(LR_RegIrqFlags, 0xFF);
}

/*****
**Name:      sx1276_7_8_LoRaEntryRx
**Function:  Entry Rx mode
**Input:     None
**Output:    None
*****/
unsigned char sx1276_7_8_LoRaEntryRx(void)
{
    unsigned char addr;

    sx1276_7_8_Config(); //setting base
parameter

    SPIWrite(REG_LR_PADAC, 0x84); //Normal and Rx
    SPIWrite(LR_RegHopPeriod, 0xFF); //RegHopPeriod NO FHSS
    SPIWrite(REG_LR_DIOMAPPING1, 0x01); //DI00=00, DI01=00,
DI02=00, DI03=01

    SPIWrite(LR_RegIrqFlagsMask, 0x3F); //Open RxDone
interrupt & Timeout
    sx1276_7_8_LoRaClearIrq();

    SPIWrite(LR_RegPayloadLength, 21); //RegPayloadLength
21byte(this register must difine when the data long of one byte in SF is 6)

    addr = SPIRead(LR_RegFifoRxBaseAddr); //Read
RxBaseAddr
    SPIWrite(LR_RegFifoAddrPtr, addr); //RxBaseAddr ->
FiFoAddrPtr
    SPIWrite(LR_RegOpMode, 0x8d); //Continuous Rx
Mode//Low Frequency Mode
    //SPIWrite(LR_RegOpMode, 0x05); //Continuous Rx
Mode//High Frequency Mode
    //SysTime = 0;
    while(1)
    {
        if((SPIRead(LR_RegModemStat)&0x04)==0x04) //Rx-on going RegModemStat
            break;
        /*if(SysTime>=3)
            return 0; //over time for

```

---

```

error*/
    }
}
/*****
**Name:      sx1276_7_8_LoRaReadRSSI
**Function:  Read the RSSI value
**Input:     none
**Output:    temp, RSSI value
*****/
unsigned char sx1276_7_8_LoRaReadRSSI(void)
{
    unsigned int temp=10;
    temp=SPIRead(LR_RegRssiValue);           //Read RegRssiValue, Rssi value
    temp=temp+127-137;                       //127:Max RSSI,
    137:RSSI offset
    return (unsigned char)temp;
}

/*****
**Name:      sx1276_7_8_LoRaRxPacket
**Function:  Receive data in LoRa mode
**Input:     None
**Output:    1- Success
             0- Fail
*****/
unsigned char sx1276_7_8_LoRaRxPacket(void)
{
    unsigned char i;
    unsigned char addr;
    unsigned char packet_size;

    if(digitalRead(dio0))//if(Get_NIRQ())
    {
        for(i=0;i<32;i++)
            RxData[i] = 0x00;

        addr = SPIRead(LR_RegFifoRxCurrentaddr);    //last packet addr
        SPIWrite(LR_RegFifoAddrPtr, addr);          //RxBaseAddr ->
        FiFoAddrPtr

        if(sx1276_7_8SpreadFactorTbl[Lora_Rate_Sel]==6)           //When
        SpreadFactor is six, will used Implicit Header mode (Excluding internal packet length)
            packet_size=21;
        else
            packet_size = SPIRead(LR_RegRxBnBytes);    //Number for received bytes
    }
}

```

---

```

    SPIBurstRead(0x00, RxData, packet_size);

    sx1276_7_8_LoRaClearIrq();
    for(i=0;i<17;i++)
    {
        if(RxData[i]!=sx1276_7_8Data[i])
            break;
    }
    if(i>=17) //Rx success
        return(1);
    else
        return(0);
}
else
    return(0);
}

/*****
**Name:      sx1276_7_8_LoRaEntryTx
**Function:  Entry Tx mode
**Input:     None
**Output:    None
*****/
unsigned char sx1276_7_8_LoRaEntryTx(void)
{
    unsigned char addr,temp;

    sx1276_7_8_Config(); //setting base
parameter

    SPIWrite(REG_LR_PADAC, 0x87); //Tx for 20dBm
    SPIWrite(LR_RegHopPeriod, 0x00); //RegHopPeriod
NO FHSS
    SPIWrite(REG_LR_DIOMAPPING1, 0x41); //DI00=01, DI01=00,
DI02=00, DI03=01

    sx1276_7_8_LoRaClearIrq();
    SPIWrite(LR_RegIrqFlagsMask, 0xF7); //Open TxDone
interrupt
    SPIWrite(LR_RegPayloadLength, 21); //RegPayloadLength
21byte

    addr = SPIRead(LR_RegFifoTxBaseAddr); //RegFiFoTxBaseAddr
    SPIWrite(LR_RegFifoAddrPtr, addr); //RegFifoAddrPtr

```

---

```

//SysTime = 0;
while(1)
{
    temp=SPIRead(LR_RegPayloadLength);
    if(temp==21)
    {
        break;
    }
    /*if(SysTime>=3)
        return 0;*/
}
}

/*****
**Name:      sx1276_7_8_LoRaTxPacket
**Function:  Send data in LoRa mode
**Input:     None
**Output:    1- Send over
*****/
unsigned char sx1276_7_8_LoRaTxPacket(void)
{
    unsigned char TxFlag=0;
    unsigned char addr;

    BurstWrite(0x00, (unsigned char *)sx1276_7_8Data, 21);
    SPIWrite(LR_RegOpMode, 0x8b);          //Tx Mode
    while(1)
    {
        if(digitalRead(dio0))//if(Get_NIRQ())           //Packet send over
        {
            SPIRead(LR_RegIrqFlags);
            sx1276_7_8_LoRaClearIrq();                //Clear irq
            sx1276_7_8_Standby();                      //Entry Standby
mode
            break;
        }
    }
}

/*****
**Name:      sx1276_7_8_ReadRSSI
**Function:  Read the RSSI value
**Input:     none
**Output:    temp, RSSI value
*****/

```

---

```

unsigned char sx1276_7_8_ReadRSSI(void)
{
    unsigned char temp=0xff;

    temp=SPIRead(0x11);
    temp>>=1;
    temp=127-temp;           //127:Max RSSI
    return temp;
}

/*****
**Name:      sx1276_7_8_Config
**Function:  sx1276_7_8 base config
**Input:     mode
**Output:    None
*****/
void sx1276_7_8_Config(void)
{
    unsigned char i;

    sx1276_7_8_Sleep();           //Change modem mode
    Must in Sleep mode
    for(i=250;i!=0;i--);          //Delay

    delay(15);

    //lora mode
    sx1276_7_8_EntryLoRa();
    //SPIWrite(0x5904); //?? Change digital regulator form 1.6V to 1.47V: see errata
    note

    BurstWrite(LR_RegFrMsb, sx1276_7_8FreqTbl[Freq_Sel], 3); //setting frequency
    parameter

    //setting base parameter
    SPIWrite(LR_RegPaConfig, sx1276_7_8PowerTbl[Power_Sel]); //Setting
    output power parameter

    SPIWrite(LR_RegOcp, 0x0B); //RegOcp, Close Ocp
    SPIWrite(LR_RegLna, 0x23); //RegLna, High & LNA
    Enable

    if(sx1276_7_8SpreadFactorTbl[Lora_Rate_Sel]==6) //SFactor=6
    {
        unsigned char tmp;

```

---

```
SPIWrite(LR_RegModemConfig1, ((sx1276_7_8LoRaBwTbl[BandWide_Sel]<<4)+(CR<<1)+0x01
)); //Implicit Enable CRC Enable(0x02) & Error Coding rate 4/5(0x01), 4/6(0x02),
4/7(0x03), 4/8(0x04)
```

```
SPIWrite(LR_RegModemConfig2, ((sx1276_7_8SpreadFactorTbl[Lora_Rate_Sel]<<4)+(CRC<
<2)+0x03));
```

```
    tmp = SPIRead(0x31);
    tmp &= 0xF8;
    tmp |= 0x05;
    SPIWrite(0x31, tmp);
    SPIWrite(0x37, 0x0C);
}
else
{
```

```
SPIWrite(LR_RegModemConfig1, ((sx1276_7_8LoRaBwTbl[BandWide_Sel]<<4)+(CR<<1)+0x00
)); //Explicit Enable CRC Enable(0x02) & Error Coding rate 4/5(0x01), 4/6(0x02),
4/7(0x03), 4/8(0x04)
```

```
SPIWrite(LR_RegModemConfig2, ((sx1276_7_8SpreadFactorTbl[Lora_Rate_Sel]<<4)+(CRC<
<2)+0x03)); //SFactor & LNA gain set by the internal AGC loop
}
```

```
    SPIWrite(LR_RegSymbTimeoutLsb, 0xFF); //RegSymbTimeoutLsb
Timeout = 0x3FF (Max)
```

```
    SPIWrite(LR_RegPreambleMsb, 0x00); //RegPreambleMsb
    SPIWrite(LR_RegPreambleLsb, 12); //RegPreambleLsb
8+4=12byte Preamble
```

```
    SPIWrite(REG_LR_DIOMAPPING2, 0x01); //RegDioMapping2
DI05=00, DI04=01
```

```
    sx1276_7_8_Standby(); //Entry standby
mode
}
```

```
void setup() {
    // put your setup code here, to run once:
    pinMode(led, OUTPUT);
    pinMode(nsel, OUTPUT);
}
```



---

```
pinMode(sck, OUTPUT);
pinMode(mosi, OUTPUT);
pinMode(miso, INPUT);
pinMode(reset, OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    mode = 0x01;//lora mode
    Freq_Sel = 0x00;//433M
    Power_Sel = 0x00;//
    Lora_Rate_Sel = 0x06;//
    BandWide_Sel = 0x07;
    Fsk_Rate_Sel = 0x00;

    sx1276_7_8_Config();//
    sx1276_7_8_LoRaEntryRx();

    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(500); // wait for a second
    digitalWrite(led, LOW); // turn the LED on (HIGH is the voltage level)
    delay(500); // wait for a second

    while(1)
    {
        //Master
        digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
        sx1276_7_8_LoRaEntryTx();
        sx1276_7_8_LoRaTxPacket();
        digitalWrite(led, LOW); // turn the LED on (HIGH is the voltage level)
        sx1276_7_8_LoRaEntryRx();
        delay(2000);

        /* if(sx1276_7_8_LoRaRxPacket())
        {
            digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
            delay(500);
            digitalWrite(led, LOW); // turn the LED on (HIGH is the voltage level)
            delay(500);
        }*/

        //slaver
        /*if(sx1276_7_8_LoRaRxPacket())
        {
```

---

```
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(500);              // wait for a second
    digitalWrite(led, LOW);  // turn the LED on (HIGH is the voltage level)
    delay(500);              // wait for a second
    sx1276_7_8_LoRaEntryRx();
    /*digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)

    sx1276_7_8_LoRaEntryTx();
    sx1276_7_8_LoRaTxPacket();
    digitalWrite(led, LOW); // turn the LED on (HIGH is the voltage level)*
    sx1276_7_8_LoRaEntryRx();*/
    //}
  }
}
```

WWW.DORJI.COM