LoRa Communication Example

This chapter will guide the user to carry out a pair of LoRa transmitting and receiving communication experiment through HopeDuino. LoRa is a unique spread spectrum technology of Semtech. HopeRF uses it to design a series of ultra high performance wireless transceiver module, with +20dBm transmit power, -139dBm ultra high sensitivity and link budget up to 159dB. The distance is 3 times of the conventional equivalent power transceiver module (test in the same indicators and the environment). The specific models are RFM92, RFM95, RFM96, RFM98, etc.

1. Tools and software needed to be prepared

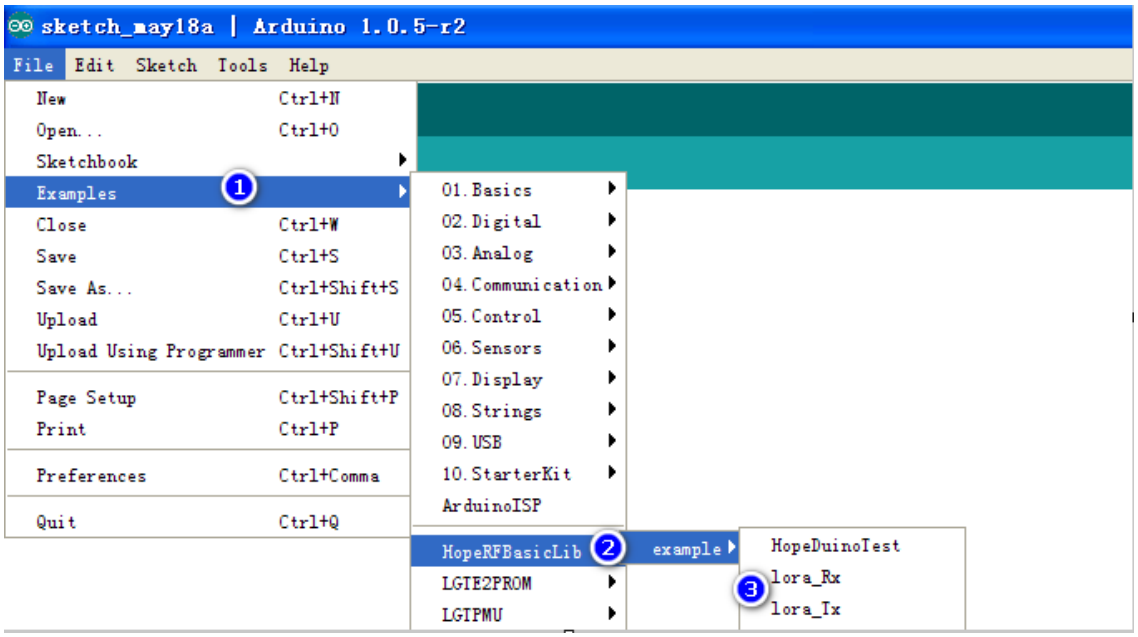
- Arduino IDE version 1.0.5
- HopeDuino board (two pieces)
  (If you have not used the HopeDuino board, please refer to the 《AN0002-HopeDuino Platform Construction Guideline》)
- USB cable (Type A to Type B)
- Module based on RF9x chip design (or module RFM9x).

RFM9x

2. Hands-on experiment

- The two RFM9x modules (with conversion board) are inserted into the corresponding HopeDuino board.
- Connect the two HopeDuino boards to PC with USB cable.
- Open Arduino IDE interface, Click 【File】→【Examples】→【HopeRFBasicLib】→【example】→【lora_Tx】, as shown below.
- Open Arduino IDE interface, Click 【File】→【Examples】→【HopeRFBasicLib】→【example】→【lora_Rx】, as shown below.

⚠ Notice: You couldn't find [HopeRFBasicLib] in [Examples] because you didn't install the HSP provided by HopeRF. Please refer to 《AN0002-HopeDuino Platform Construction Guideline》



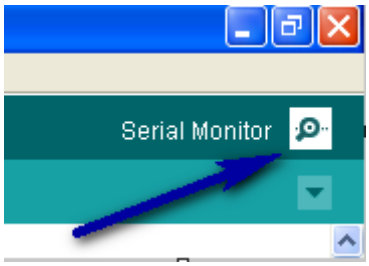
- At this time the Tx program and Rx program have been opened, please compile the download programs according to the corresponding COM port.

Notice:

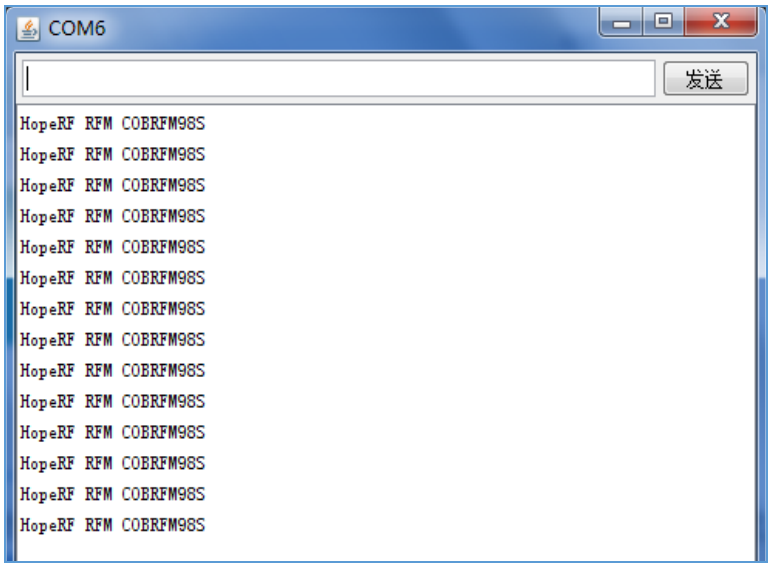
1. Do not know how to compile the download code, please refer to 《AN0002-HopeDuino Platform Construction Guideline》

2. HopeDuino platform support multiple boards connected to the same PC. But you need to modify the parameters manually. So you need to pay special attention to which COM port when you download the program every time. COM port is in the lower right corner of Arduino interface, as shown below.

- After the two programs are downloaded, the Tx board will transmit a packet of data through module RFM9x. The Rx board will receive a packet of data through module RFM9x periodically and upload the data to PC through UART (USB). At this point, you can set the COM of Arduino IDE as the port connected with Rx board. Open the “Serial Monitor” , as shown below.



- Click the “Serial Monitor”, pop up the serial port assistant interface, as shown below. Window will display the received data message.



Notice:

1. The receiving program enables UART library function. On the description of library function UART, please refer to the "HopeDuino_UART" library file. It is also stored in the HopeRFLib.

3. Program Explanation

- lora_Tx.ino Case explanation

```
#include <HopeDuino_LoRa.h>                    // Call the corresponding library file.
```

```
loraClass radio;                                                    //Define variable radio for LoRa
byte str[21] = {'H','o','p','e','R','F',' ','R','F','M',' ','C','O','B','R','F','M','9','8','S'};
                                                                    //Message to be transmitted
void setup()
{
  radio.Modulation          = LORA;          //Modulation mode is LoRa
radio.COB                   = RFM98;         //Module is RFM98
radio.Frequency             = 434000;        // Target frequency is 434MHz
radio.OutputPower           = 17;                  //Output power is 17dBm
radio.PreambleLength = 16;                   //Preamble length is 16 bytes
radio.FixedPktLength    = false;             //Message length is variable
radio.PayloadLength       = 21;              //Message length is 21 bytes
radio.CrcDisable            = true;                //Disable CRC for true

  radio.SFSel               = SF9;                 //Spread spectrum factor is 9
radio.BWSel                 = BW125K;              //Spread spectrum transmitting bandwidth is 125 KHz.
radio.CRSel                 = CR4_5;         //Coding rate is 4/5

  radio.vInitialize();                       //Initialize radio
radio.vGoStandby();                                 // Enter standby mode
}

void loop()
{
  radio.bSendMessage(str, 21);                      //Transmit a packet of message every one second
delay(1000);
}
```

➢ lora_Rx.ino Case explanation

```
#include <HopeDuino_LoRa.h>          // Call the corresponding library file.
#include <HopeDuino_UART.h>           //Calling UART is added because of using UART.

loraClass radio;                                    //Define variable radio for LoRa
uartClass uart;                              //Define variable uart for UART
byte getstr[21];                                    //Define pending data buffer

void setup()
{
  radio.Modulation          = LORA;          //Modulation mode is LoRa
radio.COB                   = RFM98;         //Module is RFM98
radio.Frequency             = 434000;        //Target frequency is 434MHz
radio.OutputPower           = 17;            //Output power is 17dBm
radio.PreambleLength = 16;                   //Preamble length is 16 bytes
```

```
radio.FixedPktLength = false;          //Message length is variable
radio.PayloadLength    = 21;           //Message length is 21 bytes.
radio.CrcDisable       = false;        //Disable CRC is false

 radio.SFSel             = SF9;              //Spreading factor is 9
 radio.BWSel              = BW125K;     //Spreading transmitting bandwidth is 125 KHz.
 radio.CRSel              = CR4_5;      //Coding rate is 4/5


 radio.vInitialize();                   //Initialize radio
radio.vGoRx();                          // Enter receiving mode
uart.vUartInit(9600, _8N1);          //Initialize UART, parameters are 9600 baud rate and 8N1 format.
}

void loop()
{
 if(radio.bGetMessage(getstr)!=0)       //Check radio whether to receive data function,
                                        //analyze data received.

    {
    uart.vUartPutNByte(getstr, 21);     // Output the received data to PC via UART
    uart.vUartNewLine();                //UART newline is easy to read.
    }
}
```

4. LoRa Library Function Description

"LoRa.h"and"LoRa.cpp"library files are stored in Arduino IDE files \ libraries \ HopeRFLib.

- **FreqStruct**

  Type: Union type

  Function: Define frequency register for LoRa chip (module)

  内容：Freq, long integer, 4 bytes, frequency value;

  FreqL，byte, low 8 bit from splitting Freq value is [0:7]

  FreqM，byte, mid 8 bit from splitting Freq value is [8:15]

  FreqH，byte, high 8 bit from splitting Freq value is [16:23]

  FreqX，byte, redundancy, rounding up 4 bytes, no meaning

- **modulationType**

  Type: Enumeration type

  Function: Select modulation mode

  Contents: OOK、FSK、GFSK、LORA

  OOK——On-Off-Key is ASK modulation, a special case of ASK modulation

  FSK——Frequency-Shift-Key, relative to the ASK has a stronger anti interference effect. But the current is larger than the ASK under the same power.

  GFSK——FSK modulation with Gauss filter

- **moduleType**
  Type: Enumeration type
  Function: Select module type
  Contents: RFM92, RFM93, RFM95, RFM96, RFM97, RFM98

- **sfType**
  Type: Enumeration type
  Function: Define spreading factor in the LoRa mode.
  Contents: SF6, SF7, SF8, SF9, SF10, SF11, SF12

- **bwType**
  Type: Enumeration type
  Function: Define spreading transmitting bandwidth in the LoRa mode.
  Contents: BW62K, BW125K, BW250K, BW500K

- **crType**
  Type: Enumeration type
  Function: Define spreading coding rate in the LoRa mode.
  Contents: CR4_5, CR4_6, CR4_7, CR4_8

- **Modulation**
  Type: Modulation type
  Function: Select one from OOK、FSK、GFSK and LoRa in the modulation and demodulation system.

- **COB**
  Type: Module type
  Function: Define the module type, COB represents Chip-On-Board, select one of RFM92, RFM93, RFM95, RFM96, RFM97 and RFM98.

- **Frequency**
  Type: lword type
  Function: working frequency, the unit is KHz, for example: Frequency = 433920， indicates 433.92MHz.

- **SymbolTime**
  Type: lword type
  Function: working rate, the unit is ns, for example: SymbolTime = 416000，indicates each symbol is 416us, that is 2.4kbps.

- **Deviation**
  Type: lword type(unsigned long)
  Function: frequency deviation for FSK and GFSK transmitting, the unit is KHz, for example: Deviation=45,

indicates the frequency deviation is 45KHz.

- **BandWidth**

  Type: word type (unsigned int)

  Function: Target receiver bandwidth for reception, the unit is KHz, for example: BandWidth = 100， indicates the receiver bandwidth is 100KHz.

- **OutputPower**

  Type: unsigned char

  Function: output power, the range is 2~20, the unit is dBm, for example: set is 10 on behalf of 10 dBm.

- **PreambleLength**

  Type: word type (unsigned int)

  Function: preamble length for transmitting, the unit is byte.

- **CrcDisable**

  Type: bool type

  Function: Select whether the data package has CRC function, set true to disable CRC function, set false to enable CRC function.

- **FixedPktLength**

  Type: bool type

  Function: Define the data packet length is fixed or variable, set true to represent the fixed packet length, set false to represent the variable length.

- **SyncLength**

  Type: byte

  Function: In wireless packet format, synchronous word length, the setting range is 1~8 bytes. Don't set to 0 bytes.

- **SyncWord[8]**

  Type: byte array

  Function: In setting packet format, synchronous word contents need to be consistent with SyncLength settings (length).

- **PayloadLength**

  Type: byte

  Function: In fixed packet length mode, define the length of the fixed packet

- **SFSel**

  Type: sfType type

  Function: Set the spreading factor in the LoRa mode, select one of SF6, SF7, SF8, SF9, SF10, SF11 and SF12.

- **BWSel**

Type: bwType type

Function: Set the transmitting bandwidth in the LoRa mode. Select one of BW62K, BW125K, BW250K and BW500K

- **CRSel**

  Type: crType type

  Function: Set coding rate in the LoRa mode, select one of CR4_5, CR4_6, CR4_7 and CR4_8.

- **vInitialize**

  Type: function

  Input: none

  Output: none

  Function: Initialize module (chip), applicable to RFM98 module, call at the start of the program. Before the call, the above related variables are set to complete. After the initialization function configuration is completed (containing call vConfig function), let the module (chip) into the Standby state, that is, non - transmitting, non - receiving, non - sleeping.

- **vConfig**

  Type: function

  Input: none

  Output: noen

Function: Configure parameters to the module (chip), suitable for the occasion needs to re configure the parameters in the working process. The same need to complete the associated variables before the call. If the associated variables set up, follow-up without modification, only to re configure the parameter, you can call it directly. If you need to switch frequency etc. in the working process, need to re modify the relevant parameters，and then call them again. After the call is completed, you need to use the mode switching function, so that let the chip work accurately in the specific mode. The mode switching functions are vGoRx、vGoStandby and vGoSleep etc.

- **vGoRx**

  Type: function

  Input: none

  Output: none

  Function: Configure module (chip) into the receiving mode

- **vGoStandby**

  Type: function

  Input: none

  Output: none

  Function: Configure module (chip) into the standby mode

- **vGoSleep**

  Type: function

  Input: none

  Output: none

Function: Configure module (chip) into the sleep mode

- **bSendMessage**

  Type: function

  Input: **msg[ ]**，unsigned char, the calling entrance (pointer) of array to be transmitted.

  length，unsigned char, the length of array to be transmitted, the unit is byte.

  Output: bool type, true indicates the transmitting is successful, false indicates the transmitting is failure, such as: push over time, etc.

  Function: transmit the data only once (one frame), return to standby mode automatically after completion of the transmission.

- **bGetMessage**

  Type: function

  Input: msg[ ]，unsigned char, the calling entrance (pointer) of array to be received.

  Output: Returns the length of the received data, 0 indicates that the data is not received;

  Function: check whether to receive data. The object is the IO state of the chip output. If you do not receive the data, return 0; if you receive the data, return the received data length. After the completion of receiving, the module (chip) is still in the receiving state.

5. Pin Assignment Table:

HopeDuino	MCU	RF9x
13	PB5	SCK
12	PB4	MISO
11	PB3	MOSI
10	PB2	nCS
9	PB1	POR
8	PB0	DIO0
7	PD7	DIO1（jumper）
6	PD6	DIO2（jumper）
5	PD5	DIO3（jumper）
4	PD4	DIO4（jumper）

6. Version Records:

Version	Revised Contents	Date
1.0	Initial version	2016-03-31
1.1	Revise text bug, add watermarks, program explanations and descriptions	2016-04-06