RF67 Communication Example

This chapter will guide the user to carry out RFM67 and RFM219 transmitting and receiving communication experiment through HopeDuino, at the same time display RF chip commonly used data message structure. RFM67 is a transmitting module based on the RF67 transmitting chip designed by HopeRF, with +17dBm transmit power, supports Sub-G applications.

1. Tools and software needed to be prepared

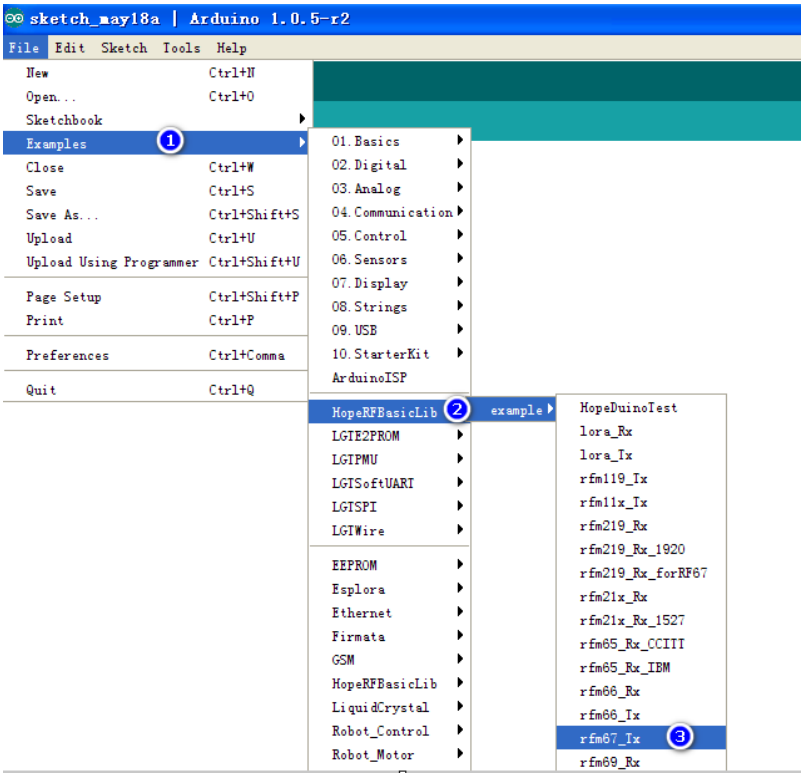
- Arduino IDE version 1.0.5
- HopeDuino board (two pieces)
  (If you have not used the HopeDuino board, please refer to the 《AN0002-HopeDuino Platform Construction Guideline》)
- USB cable (Type A to Type B)
- CMOSTEK RFPDK software
- RFM67 module (or module based on RF67 chip design)
- RFM219 module (or module based on CMT2219A design)

RFM67

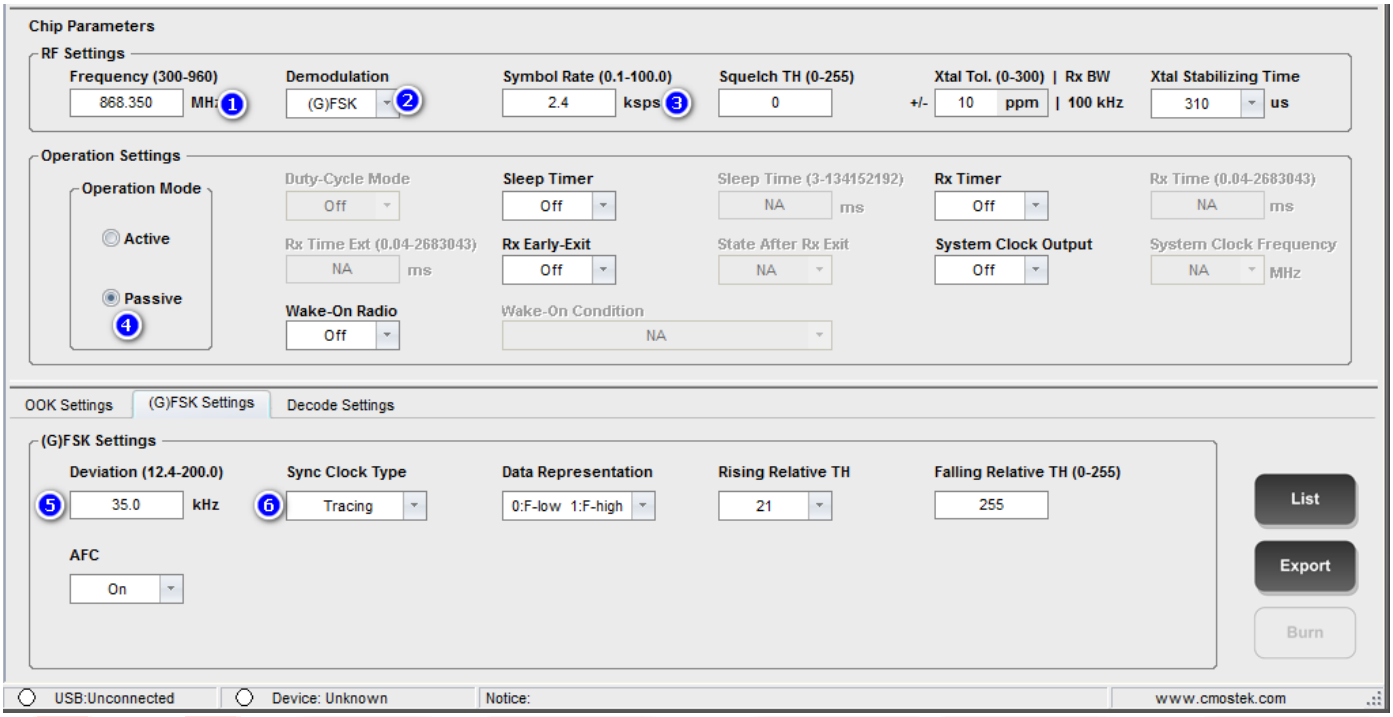
2. Hands-on experiment

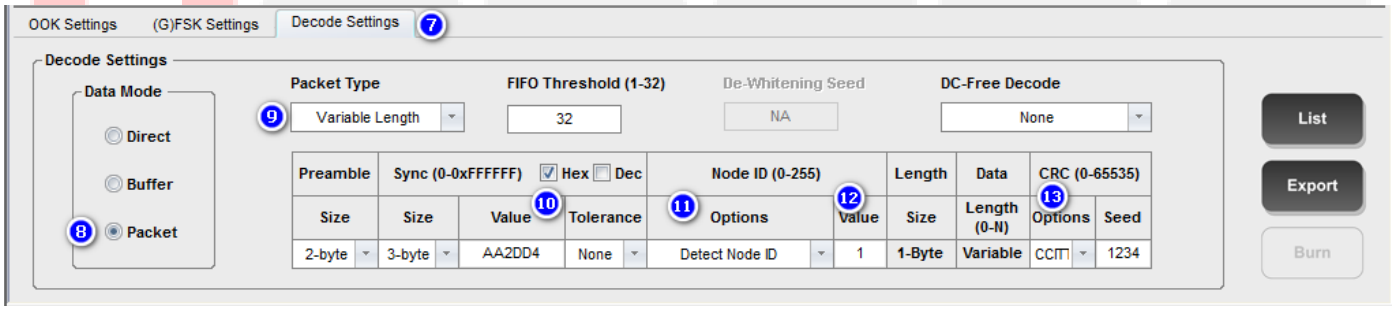
- Inset RFM67 and RFM219 modules (with conversion board) into the corresponding HopeDuino board.
- Connect the two HopeDuino boards to PC with USB cable.
- Open Arduino IDE interface, Click 【File】→【Examples】→【HopeRFBasicLib】→【example】→【rfm67_Tx】, as shown below.

⚠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》.



- Open Arduino IDE interface, Click【File】→【Examples】→【HopeRFBasicLib】→【example】→【rfm219_Rx_forRF67】, as shown above.
- The configuration parameters on the rfm219_Rx_forRF67.ino are as follows RFPDK interface screenshot.





Notice: In order to better display the common RF chip data packet structure, this experiment increases RFM219 setting functions as follows:

（1） Become long packet format, which is the length of useful information is embedded in the message, so that the length of each message is not the same.

（2） The data packet increases the NodeID, that is, the receiving message can be filtered through the NodeID matching or not. In this paper, the set of NodeID is 1 (0x01).

（3） Increase CRC-16 at the end of data packet, which is whether the received message is correct or not by CRC-16, determine whether the message is all correct. In this paper, we set the CRC-16 polynomial to use CCITT, the starting value is 1234 (decimal).

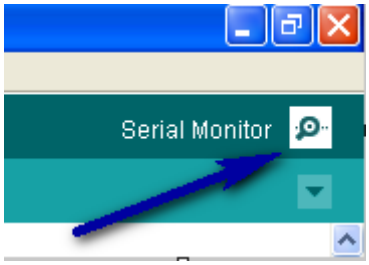
- At this time the Tx program and Rx program have been opened, please compile the download programs according to the corresponding COM port.

Notice:

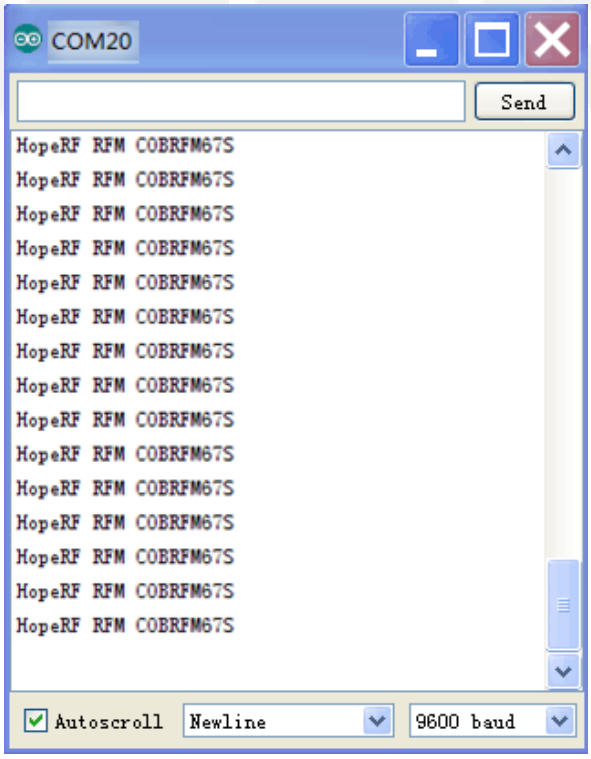
1. Do not know how to compile the download code, please refer to 《AN0002-HopeDuino Platform Construction Guideline》

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 RFM67. The Rx board will receive a packet of data through module RFM219 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.

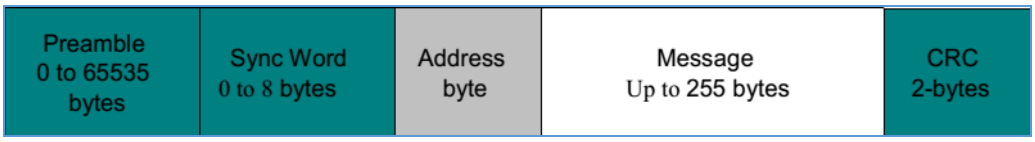


⚠ Notice:

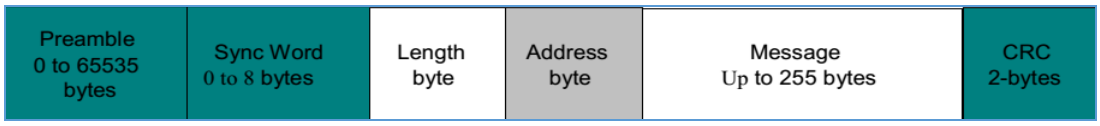
1. The receiving program enables UART library function. On the description of library function UART, please refer to the "HopeDuino_UART" library file. It is also stored in the HopeRFLib.

3. Brief introduction of common RF chip message:

- Common message definitions are: fixed length messages and variable length messages, as shown below:



Fixed length message



Variable length message

- Whether it is "fixed length message" or "variable length message", must have three parts: Preamble, SyncWord, Message:

(1) Preamble is popular as a boot code. There are 0xAA or 0x55 compositions; it is 50% square wave from the timing analysis. Although there is no real meaning, it will help the receiver side of the chip through it to complete the signal tracking, recognition rate, as well as AFC and other advanced features. So Preamble is very important in a complete data message.

(2) SyncWord is popular as synchronization word; of course, it can be more than one byte from the length. For the purposes of this experiment using CMT2219A, this synchronization word can be defined from 1 to 4 bytes. Since the front Preamble is square wave, if there is no one synchronous identification mark, the receiver is unable to determine the effective information from that moment. And the synchronization word is to play the role, after the word is the beginning of the useful signal.

(3) Message is the effective data, can also become a Payload. The vast majority of RF chips all support the three basic parts of the message format, which is the most basic, the most common, at the same time, it is also the most basic mode of interactive communication between different manufacturers.

- Corresponding to the above three parts, there are three optional parts - length, node address, CRC.

(1) Length is the length of the effective content of the message, it is optional. If the length is embedded in the message, it is defined as variable length message. If the length is not embedded in the message, both sides are aware of the other side of the effective content length. This is what is called a fixed length message. At the same time, length refers to the length of the message generally. In the case of the node address is valid, it will increase 1 bytes. Generally it does not contain the length of its own, CRC, Preamble, and SyncWord.

(2) Node address can be used as a filter. Maybe the user asks: is this not the same effect with the SyncWord? Yes, the efficacy is similar, but SyncWord is suitable for the identification of different networks. And for a network, the need to distinguish the identity of the different nodes within the network, the use of node address is more advantage. In addition, the node address position and length value can be mutually. Just as the CMT2219A, which is used in this experiment, is the node address in the front, the length value in the rear.

(3) CRC is the check information at the end of the message, uses CRC-16 generally. Common CCITT and IBM are two different polynomials. This is a follow-up to the documentation.

4. Program Explanation

➢ rfm67_Tx.ino Case explanation

```
#include <HopeDuino_RFM67.h>              // Call the corresponding library file.
rf67Class radio;                          // Define variable radio for rf67.

byte str[21] = {'H','o','p','e','R','F',' ','R','F','M',' ','C','O','B','R','F','M','6','7','S'};
                                                  //Message to be transmitted

void setup()
{
  radio.Modulation        = FSK;          //Modulation mode is FSK
  radio.Frequency         = 868350;             //Target frequency is 868.35MHz
  radio.OutputPower       = 10+13;          //Output power is 10dBm
  radio.PreambleLength = 16;              //Preamble length is 16 bytes
radio.FixedPktLength = false;           //Message length is variable
radio.PayloadLength    = 21;            //Payload length is 21 bytes
radio.CrcDisable        = false;        //Enable CRC,
                                                  //also enable CRC for the CMT2219A configuration.
  radio.CrcInverse       = false;         // CRC calculation results do not take the reverse
radio.CrcMode             = false;       // CRC uses CCITT polynomials
radio.CrcSeed           = 1234;          // CRC initial value is 1234 (decimal),
                                                  //it is also corresponding with the CMT2219A configuration
  radio.NodeDisable      = false;         //Enable node function
radio.NodeBefore        = true;          //Node is before length
radio.NodeAddr            = 0x01;        //Node address is 0x01

  radio.SymbolTime       = 416000;          //Symbol time is 2.4Kbps
radio.Deviation          = 35;            // Frequency deviation is 35 KHz
radio.SyncLength        = 3;             // Sync length is 3 bytes
radio.SyncWord[0]       = 0xAA;          //Sync word is 0xAA2DD4
radio.SyncWord[1]       = 0x2D;
  radio.SyncWord[2]      = 0xD4;
  radio.vInitialize();                  // Initialize radio
  radio.vGoStandby();                   // Enter standby mode
}

void loop()
{
  radio.bSendMessage(str, 21);          // Transmit a packet of message every one second,
  //radio.vGoStandby();                 //enter sleep mode after transmitting
  radio.vGoSleep();                     // (users can change to standby mode according to the need).
  delay(1000);
}
```

➢ rfm219_Rx_forRF67.ino Case Explanation

```
#include <HopeDuino_CMT2219A.h>          //Call CMT2219A 和 UART library files
#include <HopeDuino_UART.h>
cmt2219aClass radio;                     //Define variable radio for CMT2219A
uartClass uart;                          // Define variable uart for UART
byte getstr[21];                         // Define pending data buffer

byte CfgTbl[62] = {                      // CMT2219A configuration parameters, derived by the RFPDK.
                    0x72,                //  Mode                        = Advanced
                    0x42,                //  Part Number                 = CMT2219A
                    0x44,                //  Frequency                   = 868.350 MHz
                    0x15,                //  Demodulation                = (G)FSK
                    0x0D,                //  Symbol Rate                 = 2.4 ksps
                    0x63,                //  Xtal Tolerance              = +/- 10 ppm
                    0x9A,                //  Xtal Stabilizing Time       = 310 us
                    0x80,                //  Squelch TH                  = 0
                    0xC6,                //  Sleep Timer                 = Off
                    0x53,                //  Sleep Time                  = NA
                    0x01,                //  Rx Timer                    = Off
                    0x00,                //  Rx Time                     = NA
                    0x62,                //  Rx Time Ext                 = NA
                    0x1E,                //  Rx Early Exit               = Off
                    0x00,                //  State After Rx Exit         = NA
                    0x10,                //  System Clock Output         = Off
                    0x84,                //  System Clock Frequency      = NA
                    0x14,                //  Wake-On Radio               = Off
                    0xE0,                //  Wake-On Condition           = NA
                    0x00,                //  Demod Method                = NA
                    0x27,                //  Fixed Demod TH              = NA
                    0x9F,                //  Peak Drop                   = NA
                    0x00,                //  Peak Drop Step              = NA
                    0xD4,                //  Peak Drop Rate              = NA
                    0x2D,                //  Deviation                   = 35.0 kHz
                    0xAA,                //  Sync Clock Type             = Tracing
                    0x00,                //  Data Representation         = 0:F-low    1:F-high
                    0x38,                //  Rising Relative TH          = 21
                    0xC5,                //  Falling Relative TH         = 255
                    0x01,                //  AFC                         = On
                    0x51,                //  Data Mode                   = Packet
                    0x21,                //  Packet Type                 = Variable Length
                    0x07,                //  FIFO Threshold              = 32
                    0x84,                //  De-Whitening Seed           = NA
                    0x00,                //  DC-Free Decode              = None
                    0x00,                //  FILE CRC                    = 9DCC
                    0x19,
```

```
                0x00, 0x00, 0x06, 0xAC, 0xAE, 0x53, 0xD4, 0x40, 0x49, 0xFF,
                0x1D, 0x12, 0x08, 0x90, 0xFA, 0x00, 0x00, 0x40, 0xC0, 0x00,
                0x00, 0x20, 0xCA, 0x97, 0x00
            };

void setup()
{
  radio.CrcDisable        = false;                //Enable CRC
radio.FixedPktLength = false;             // Message length is variable
radio.NodeDisable       = false;                //Enable node function
radio.vInit(CfgTbl);                      //Initialize radio
radio.vGpioFuncCfg(GPIO1_INT1|GPIO2_DCLK|GPIO3_CLK|GPIO4_Dout);
                                          //Configure GPIO parameters, GPIO1 is INT1,
                                          //GPIO2 is DCLK (demodulation data synchronous clock output),
                                          //GPIO3 is CLK (clock division),
                                          //GPIO4 is DATA (demodulation data stream output).
radio.vIntSourcCfg((FIFO_WBYTE+OFFSET), 0);     //INT1 configuration is WBYTE interrupt,
                                         // Each time a byte is received to generate an interrupt signal.
radio.vEnableIntSource(0xFF);             //Enable full interrupt source
radio.vGoRx();                            //Enter receiving state
uart.vUartInit(9600, _8N1);              // Initialize UART, parameters are 9600 baud rate and 8N1 format.
}
void loop()
{
  byte length;
  length = radio.bGetMessage(getstr);   //Check radio whether to receive data function,
                                            //analyze data received.
  if(length!=0)
      {
      uart.vUartPutNByte(getstr, length);      // Output the received data to PC via UART
      uart.vUartNewLine();                     //UART newline is easy to read.
      }
}
```

5. RF67 Library Function Description

"RF67.h"and"RF67.cpp"library files are stored in Arduino IDE files \ libraries \ HopeRFLib.

- **FreqStruct**

  Type: union type

  Function: Define frequency register for LoRa chip (module).

  Contents: 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

  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.

- **Modulation**

  Type: modulation type

  Function: Define modulation mode, select one of OOK, FSK and GFSK

- **Frequency**

  Type: lword type (unsigned long)

  Function: working frequency, the unit is KHz, for example: Frequency = 433920， indicates 433.92MHz.

- **SymbolTime**

  Type: lword type (unsigned long)

  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 45 KHz.

- **OutputPower**

  Type: unsigned char

  Function: output power for transmitting, the range is -13dBm～+17dBm, for example: Set is 13+10, on behalf of 10 dBm.

- **PreambleLength**

  Type: word type (unsigned int)

  Function: preamble length for transmitting, the unit is byte, the range is 0~31.

- **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.

- **CrcMode**

  Type: bool type

Function: Select CRC-16 polynomial mode, set true to use IBM polynomial ($X^{16}+X^{15}+X^2+1$); set false to use CCITT ($X^{16}+X^{12}+X^5+1$).

- **CrcSeed**

  Type: unsigned int

  Function: the initial value of CRC-16 calculation.

- **CrcInverse**

  Type: bool type

  Function: Select whether the CRC-16 needs to inverse the results after calculation completion, set true to inverse the results; set false not to inverse the results.

- **NodeDisable**

  Type: bool type

  Function: Select whether the packet has NodeID function, set true to disable NodeID function, set false to enable NodeID function.

- **NodeAddr**

  Type: unsigned char type

  Function: Specific value of NodeID

- **NodeBefore**

  Type: bool type

  Function: Select the packet format, set true, the packet format is:
  Preamble + SyncWord + NodeID + Length + Message + CRC;
  Set false, the packet format is:
  Preamble + SyncWord + Length + NodeID + Message + CRC;
  where CRC and Length are optional.

- **FixedPktLength**

  Type: bool type

  Function: Define whether the packet is fixed length or variable length, set true to represent the fixed 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 needs to be consistent with SyncLength settings.

- **PayloadLength**

  Type: byte

Function: In fixed packet length mode, defines the length of the fixed packet.

- **vInitialize**

  Type: function

  Input: none

  Output: none

  Function: Initialize module (chip), applicable to RFM67 module, call it 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 - sleeping.

- **vConfig**

  Type: function

  Input: none

  Output: none

  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 are 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 vGoFs、vGoTx、vGoStandby and vGoSleep.

- **vGoTx**

  Type: function

  Input: none

  Output: none

  Function: Configure module (chip) into the transmitting mode

- **vGoFs**

  Type: function

  Input: none

  Output: none

  Function: Configure module (chip) into the frequency synthesis mode (No transmitting, only keep the crystal oscillator starting, the PLL is locked).

- **vGoStandby**

  Type: function

  Input: none

  Output: none

  Function: Configure module (chip) into 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.

- **vChangeFreq**

  Type: function

  Input: freq, unsigned long type, target frequency to be switched.

  Output: none

  Function: switch to target frequency.

6. Pin Assignment Table:

HopeDuino	MCU	RF67	CMT2219A
13	PB5	SCK	SCL
12	PB4	MISO	FCSB
11	PB3	MOSI	SDA
10	PB2	nCS	CSB
9	PB1	DCLK	
8	PB0	POR	GPO1
7	PD7	EOL（jumper）	GPO2（jumper）
6	PD6	DATA（jumper）	GPO4（jumper）
5	PD5	PLL_LOCK（jumper）	GPO3（jumper）
4	PD4		

7. Version Records:

Version	Revised Contents	Date
1.0	Initial version	2016-03-31
1.1	Add example explanations	2016-04-07