2.4G GFSK 双向透传模块

BMC56M001
Arduino Library V1.0.2 说明

版本：V1.10　日期：2023-07-25

www.bestmodulescorp.com

目录

简介

BMC56M001 是倍创推出的一款 2.4G GFSK 双向透传模块，使用 UART 通信方式。本文档对 BMC56M001 的 Arduino Lib 函数、Arduino Lib 安装方式进行说明；范例演示了模块搭配形成 Peer 网络拓扑或 Star 网络拓扑实现配对以及无线通信等功能。

Arduino Lib 函数

Arduino Lib 名称：BMC56M001		Lib 版本：V1.0.2
构造函数 & 初始化		
1	BMC56M001(HardwareSerial *theSerial=&Serial)	
	描述	构造函数，使用 HW Serial 接口
	参数	*theSerial：选择 HW Serial 接口 (默认 Serial 接口)
	返回值	—
	备注	—
2	BMC56M001(uint8_t rxPin, uint8_t txPin)	
	描述	构造函数，使用 SW Serial 接口
	参数	rxPin：RX 引脚，连接 BMC56M001 的 TX 引脚 txPin：TX 引脚，连接 BMC56M001 的 RX 引脚
	返回值	—
	备注	—
3	void begin(uint8_t baud=BR_9600)	
	描述	模块初始化
	参数	baud：通信速率选择 0x00 (BR_9600)：9600bps (默认) 0x01 (BR_19200)：19200bps 0x02 (BR_38400)：38400bps
	返回值	void
	备注	—
功能函数		
4	bool isPaired()	
	描述	是否配对过
	参数	—
	返回值	是否配对过状态 TRUE：已经配对过 FALSE：没配对过
	备注	(1) 此函数仅支持硬件 V1.01 及以上的版本号，V1.00 不支持 (2) 如果模块已经配对过，则不需要重新配对。可直接获取通信短地址，进行通信。 (3) 通信短地址可通过下面方式获取： Peer 模式：配对成功后，使用 getShortAddress 函数进行获取 Star 模式： Concentrator：依据上次配对顺序的短地址 (0x0001~0x0005) Node：配对成功后，使用 getShortAddress 函数进行获取

5	uint8_t writePairPackage()	
	描述	发送配对包
	参数	—
	返回值	执行情况 *
	备注	—
6	uint8_t getPairStatus()	
	描述	获取配对状态
	参数	—
	返回值	配对状态 0x00：配对中 0x01：配对成功 0x02：配对失败 0x03：配对超时
	备注	—
7	uint8_t writeRFData(uint32_t shortAddr, uint8_t len, uint8_t data[])	
	描述	发送数据包
	参数	shortAddr：短地址 len：需要发送数据的长度 data[]：需要发送的数据
	返回值	执行情况 *
	备注	发送数据的长度不能超过 32-byte
8	bool isInfoAvailable()	
	描述	判断是否接收到数据
	参数	—
	返回值	数据接收情况 false：没接收到数据 true：接收到数据
	备注	—
9	uint8_t readRFData(uint8_t rxData[], uint8_t &len)	
	描述	读取 RF 封包里面的数据
	参数	rxData[]：存储接收到的数据 &len：存储接收到数据的长度
	返回值	接收到的数据情况 0x00：不是数据包 0x01：是数据包
	备注	rxData[] 的长度建议设为 36-byte
10	uint8_t getShortAddress()	
	描述	获取通信短地址
	参数	—
	返回值	通信短地址
	备注	在配对成功后 Peer 角色或 Node 角色可获取通信短地址，用于发送数据包

11	uint8_t getRSSI()	
	描述	获取当前信号强度
	参数	—
	返回值	当前信号强度
	备注	—
12	uint8_t getPktRSSI()	
	描述	获取接收封包信号强度
	参数	—
	返回值	接收封包信号强度
	备注	—
13	uint8_t writeEEPROM(uint8_t len, uint8_t deviInfo[])	
	描述	向 EEPROM 写入数据
	参数	len：需要写入数据的长度 deviInfo[]：需要写入的数据
	返回值	执行情况 *
	备注	deviInfo[] 的长度不能超过 32-byte
14	uint8_t readEEPROM(uint8_t deviInfo[], uint8_t &len)	
	描述	读取 EEPROM 中的数据
	参数	deviInfo[]：存储获取的数据 &len：存储获取数据的长度
	返回值	执行情况 *
	备注	deviInfo[] 的长度建议设为 32-byte
15	uint8_t getFWVer(uint8_t number[])	
	描述	获取版本号
	参数	number[]：存储接收到的版本号
	返回值	执行情况 *
	备注	number[] 的长度建议设为 16-byte
16	uint8_t getSN(uint8_t id[])	
	描述	获取序列号
	参数	id[]：存储获取的序列号
	返回值	执行情况 *
	备注	id[] 的长度建议设为 4-byte
设置 & 读取函数		
17	uint8_t getDeviceRole()	
	描述	获取设备角色
	参数	—
	返回值	设备角色 0x00：Peer 角色 0x01：Node 角色 0x02：Concentrator 角色
	备注	—

18	uint8_t getMode()	
	描述	获取工作模式
	参数	—
	返回值	工作模式 0x00：深睡眠模式 0x01：睡眠模式 0x02：RX 模式 0x03：配对模式
	备注	—
19	uint8_t getChannelPtn()	
	描述	获取跳频频道
	参数	—
	返回值	跳频频道 0x00：跳频组 1 0x01：跳频组 2 ..... 0x0F：跳频组 16
	备注	—
20	uint8_t getRFPower()	
	描述	获取发射功率
	参数	—
	返回值	发射功率 0x00：-3dBm 0x01：0dBm 0x02：5dBm 0x03：7dBm
	备注	—
21	uint8_t getDataRate()	
	描述	获取空中通信速率
	参数	—
	返回值	空中通信速率 0x00：125kbps 0x01：250kbps 0x02：500kbps
	备注	—
22	uint8_t getHoppPeriod(uint8_t period[])	
	描述	获取跳频周期
	参数	period[]：存储跳频周期参数，范围：0x0002~0xFFFE 跳频周期 = 8μs* 跳频周期参数
	返回值	执行情况 *
	备注	—

23	uint8_t getBaud()	
	描述	获取通信速率
	参数	—
	返回值	通信速率 0x01：9600bps 0x02：19200bps 0x03：38400bps
	备注	—
24	uint8_t setDeviceRole(uint8_t role)	
	描述	设置设备角色
	参数	role：设备角色 0x00 (Peer)：Peer 角色 0x01 (Node_of_Star)：Node 角色 0x02 (Concentrator_of_Star)：Concentrator 角色
	返回值	执行情况 *
	备注	Peer 角色的设备可俩俩之间组成 Peer 网络拓扑进行数据交流。 Concentrator 角色的设备可与多个 (最多 5 个) Node 角色的设备组成 Star 网络拓扑进行数据交流。
25	uint8_t setMode(uint8_t mode)	
	描述	设置工作模式
	参数	mode：工作模式 0x00 (DeepSleep_Mode)：深睡眠模式 0x01 (Sleep_Mode)：睡眠模式 0x02 (Rx_Mode)：RX 模式 0x03 (Pairing_Mode)：配对模式
	返回值	执行情况 *
	备注	—
26	uint8_t setChannelPtn(uint8_t channel)	
	描述	设置跳频频道
	参数	channel：跳频频道 0x00 (ChannelG_1)：跳频组 1 0x01 (ChannelG_2)：跳频组 2 ..... 0x0f (ChannelG_16)：跳频组 16
	返回值	执行情况 *
	备注	—
27	uint8_t setRFPower(uint8_t power)	
	描述	设置发射功率
	参数	power：发射功率 0x00 (N3dBm)：-3dBm 0x01 (P0dBm)：0dBm 0x02 (P5dBm)：5dBm 0x03 (P7dBm)：7dBm
	返回值	执行情况 *
	备注	—

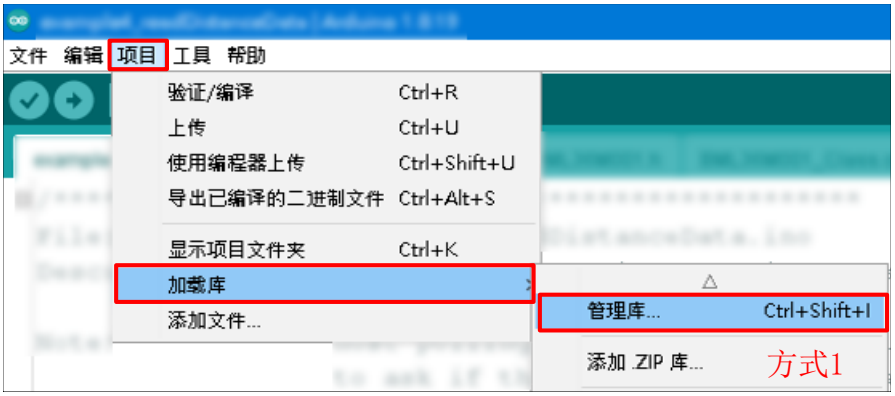
28	uint8_t setDataRate(uint8_t rate)	
	描述	设置空中通信速率
	参数	rate：空中通信速率 0x00 (DR125Kbps)：125kbps 0x01 (DR250Kbps)：250kbps 0x02 (DR500Kbps)：500kbps
	返回值	执行情况 *
	备注	—
29	uint8_t setHoppPeriod(uint8_t period[])	
	描述	设置跳频周期
	参数	period[]：跳频周期参数，范围 0x0002~0xFFFE 跳频周期 = 8μs* 跳频周期参数
	返回值	执行情况 *
	备注	—
执行情况 *：0 – 指令执行成功；1 – 指令执行失败；2 – 指令不支持；3 – 格式错误；4 – 数据太长；5 – 配对失败；6 – 配对超时；7 – 发送失败；8 – 发送成功		

Arduino Lib 下载及安装

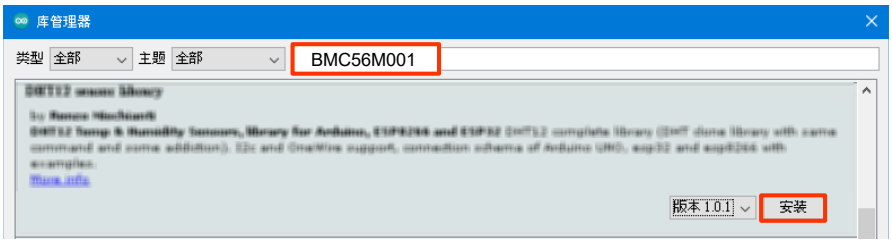
BMC56M001 Library：可参考下面两种方法安装 BMC56M001 的 Arduino Library

方式 1：搜索安装

搜索安装：Arduino IDE → 项目 → 加载库 → 管理库 → 搜索 BMC56M001 → 安装



搜索安装流程 1

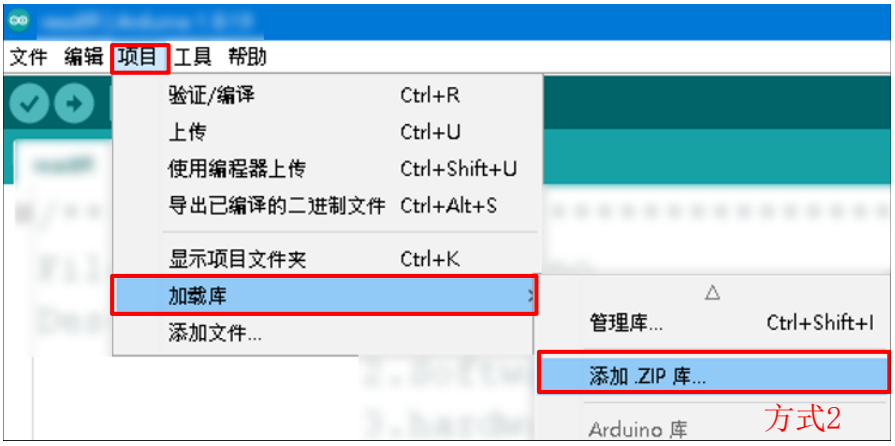


搜索安装流程 2

方式 2：添加 .ZIP 库，需提前下载 .ZIP 库

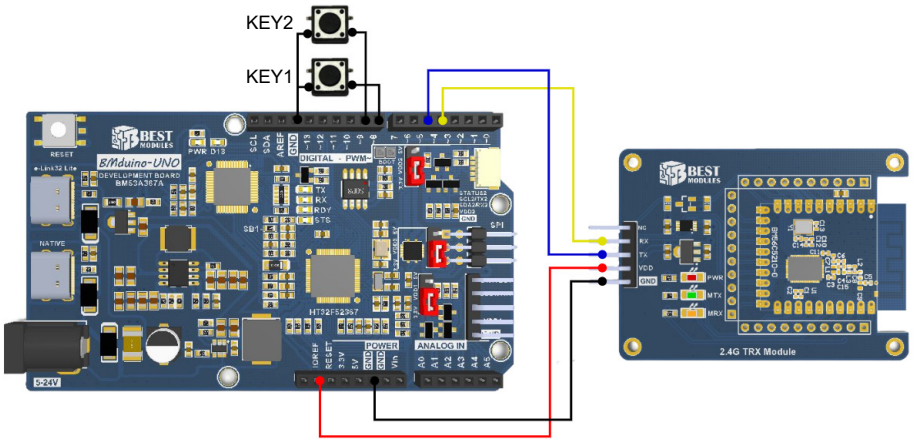
下载方法：打开倍创官方网站 (https://www.bestmodulescorp.com/bmc56m001.html#tab-product2) 文件目录下的 Arduino 范例程序 (BMC56M001 Library)。

添加 .ZIP 库：Arduino IDE → 项目 → 加载库 → 添加 .ZIP 库 ....



Arduino 范例

范例 1：Peer



实物连接示意图

范例实现功能：设置模块角色为 Peer 并且与同为 Peer 角色的模块搭配形成 Peer 网络拓扑并且进行配对以及数据交流。

两个模块可同时运用此范例搭配形成 Peer 网络拓扑并且进行配对以及数据交流。

步骤如下：

(1). Peer 端与 Peer 端配对

两个模块皆按下 KEY1 (D8)，进入配对模式，配对时间为 8 秒，此时 MRX 指示灯闪烁；任意一个模块，短按 KEY2 (D9) 会发送配对请求封包，配对成功 MRX 指示灯熄灭，并且在串口监视器上显示配对成功，若 8 秒内没有配对成功 MRX 指示灯也会熄灭，用户可再次重新配对。

(2). 接收与发送数据切换

两个模块配对成功后，都进入接收模式，用户可短按任意一模块 KEY2 (D9) 即可发送数据包。

(3). Peer 端给 Peer 端发送数据

用户可短按任意一模块的 KEY2 (D9) 来发送数据包，此时 MTX 指示灯会闪烁一次指示有数据包发出。另一模块收到发送的数据时，MRX 指示灯会闪烁一次指示有接收到数据包，并在串口监视器上显示接收到的数据。

1. 范例打开方式：Arduino IDE→ 文件 → 示例 → Lib 选择 (BMC56M001) → 选择范例 (Peer)
2. 示例说明：

a. 创建对象 & 模块初始化及配置

```
#include  "BMC56M001.h"
BMC56M001      BMC56(5,4); // TX 引脚连接开发板 D5，RX 引脚连接开发板 D4
#define    KEY1_Pin (8)       // 将 D8 与 GND 通过按键连接，此按键为 KEY1
#define    KEY2_Pin (9)       // 将 D9 与 GND 通过按键连接，此按键为 KEY2
uint8_t   Message_ShortAddr;
bool Flag_Pairing,Flag_PairSuccess;
// 发送配对包中的数据部分
uint8_t TXDATA[16] = {0},RXDATA[32] = {0};
uint8_t DATA,STATUS,len;
/******* 函数声明 *******/
uint8_t Sys_KEY(void);           // 获取按键状态
void RFMessage_Process();        // 根据按键状态执行相关动作
void Handle_RFPacket_Process(); // 获取数据包 & 获取配对回复包
void setup()
{
  /***** 按键初始化 *****/
  pinMode(KEY1_Pin, INPUT_PULLUP);
  pinMode(KEY2_Pin, INPUT_PULLUP);
  Serial.begin(115200);          // 配置串口监视器
  BMC56.begin(BR_38400);         // 初始化模块，配置通信速率
  BMC56.setDeviceRole(Peer);     // 选择设备角色
}
```

b. 根据按键状态执行配对、发送配对包、发送数据包等操作，有接收到数据时获取数据并在串口监视器上显示

```
void loop()
{
  RFMessage_Process();           // 扫描按键
  Handle_RFPacket_Process();     // 扫描是否接收到数据
}
```

c. 获取按键状态函数

```
uint8_t Sys_KEY(void)
{
  if(!digitalRead(KEY1_Pin))
  {
    delay(60);
    if(!digitalRead(KEY1_Pin))
    {
      return 0x01;
    }
  }
  if(!digitalRead(KEY2_Pin))
  {
    delay(60);
    if(!digitalRead(KEY2_Pin))
    {
      return 0x02;
    }
  }
  return 0x00;
 }
```

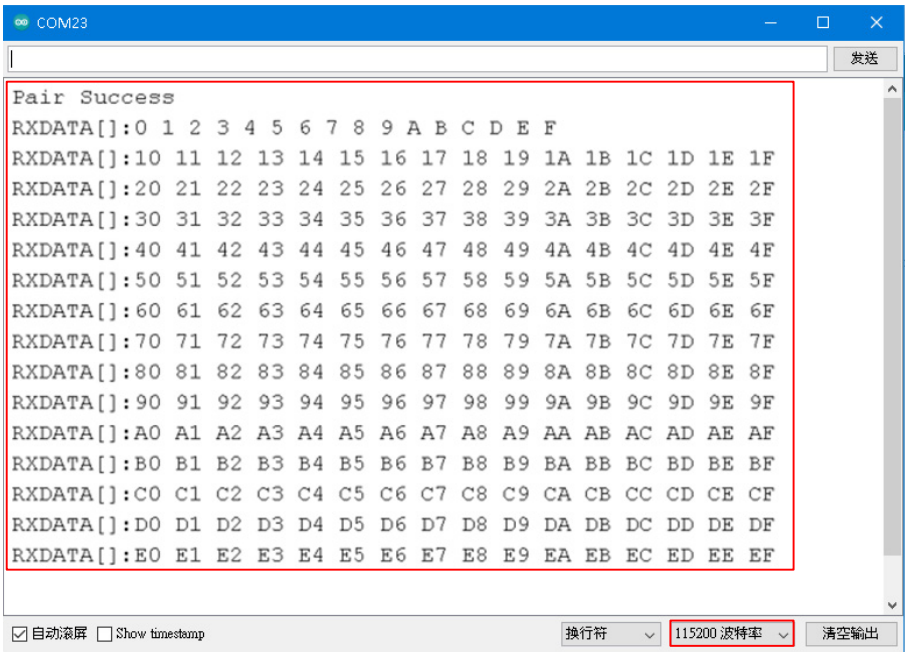
d. 根据按键状态执行相关程序

```
void RFMessage_Process()
{
  switch(Sys_KEY())
  {
    case 0x01:
    /****KEY1 按键有按下 ****/
    BMC56.setMode(Pairing_Mode);  // 进入配对模式
    Flag_Pairing = TRUE;
    Flag_PairSuccess = FALSE;
    break;
    case 0x02:
    /****KEY2 按键有按下 ****/
    if(Flag_Pairing)              // 判断是否正在配对中
    {
      BMC56.writePairPackage();   // 发送配对请求包
    }
    if(Flag_PairSuccess)          // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Message_ShortAddr,16,TXDATA); // 发送数据包
    }
    break;
  }
}
```

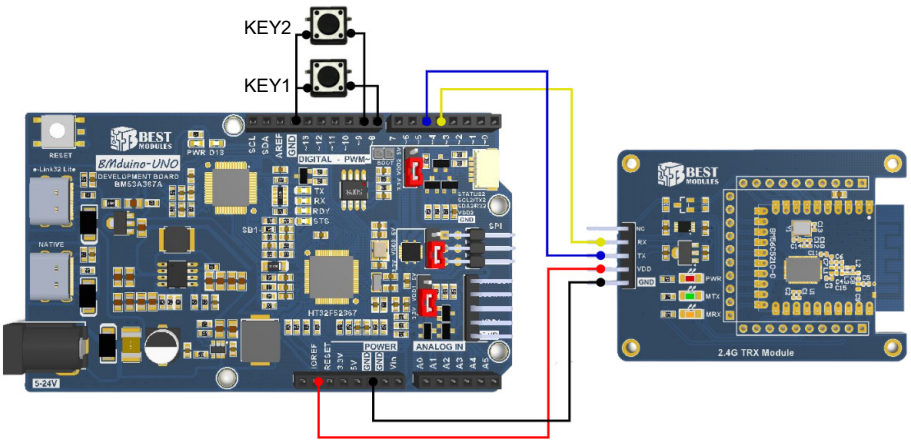
e. 接收到数据时在串口监视器上显示

```
void Handle_RFPacket_Process()
{
  if(Flag_Pairing)                          // 判断是否在配对中
  {
    STATUS = BMC56.getPairStatus();         // 获取配对情况
    if(STATUS == 1)// pairing success       // 判断是否配对成功
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = TRUE;
      Message_ShortAddr = BMC56.getShortAddress(); // 获取短地址
      BMC56.setMode(Rx_Mode);               // 进入 RX 模式
      Serial.println("Pair Success");       // 在串口监视器上显示 "Pair
                                            // Success"
    }
    if(STATUS == 2)                         // 判断是否配对失败
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
    if(STATUS == 3)                         // 判断是否配对超时
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
  }
  if(Flag_PairSuccess)                      // 判断是否已经配对成功
  {
    if(BMC56.isInfoAvailable())             // 判断是否有数据待读取
    {
      STATUS = BMC56.readRFData(RXDATA,len);        // 读取数据
      if(STATUS == 1)                       // 判断读取的数据是否为数据包
      {
        Serial.print("RXDATA[]:");          // 串口监视器上显示读取的数据
        for(uint8_t temp=0;temp<len;temp++)
        {
          Serial.print(RXDATA[temp],HEX);
          Serial.print(" ");
        }
        Serial.println(" ");
      }
    }
  }
}
```

3. 打开串口监视器，波特率选择 115200；串口监视器显示如下



范例 2：Node



实物连接示意图

范例实现功能：设置模块角色为 Node 并且与 Concentrator 角色的模块搭配形成 Star 网络拓扑并且进行配对以及数据交流。

此范例需搭配范例 Concentrator 使用。

步骤如下：

(1). Node 端与 Concentrator 端配对

两个模块皆按下 KEY1 (D8)，进入配对模式：配对时间为 8 秒，此时 MRX 指示灯闪烁；Node 端短按 KEY2 (D9) 会发送配对请求封包，配对成功 MRX 指示灯熄灭，并且在 Node 端与 Concentrator 端的串口监视器上显示配对成功，若 8 秒内没有配对成功 MRX 指示灯也会熄灭，用户可再次重新配对。

(2). 接收与发送数据切换

Node 端与 Concentrator 端配对成功后，都进入接收模式，用户可短按任意 Node 端 KEY2 (D9) 来发送数据包给配对好的 Concentrator 端；或短按 Concentrator 端 KEY2~KEY6 (D9~D13) 来发送数据包给配对好之不同 Node。

(3). Node 端给 Concentrator 端发送数据

用户可短按 Node 端 KEY2 (D9) 来发送数据封包，此时 Node 端 MTX 指示灯会闪烁一次指示有数据包发出。Concentrator 端接收到数据时 MRX 指示灯会闪烁一次指示有接收到数据包，并在串口监视器上显示接收到的数据。

(4). Concentrator 端给 Node 端发送数据

用户可短按 Concentrator 端 KEY2 (D9) 来发送数据封包，此时 Concentrator 端 MTX 指示灯会闪烁一次，指示有数据包发出。Node 端接收到数据时 MRX 指示灯会闪烁一次指示有接收到数据包，并在串口监视器上显示接收到的数据。若有超过一个以上 Node，可按照配对顺序短按 Concentrator 端 KEY2~KEY6 (D9~D13) 分别对不同 Node 发送数据。

1. 范例打开方式：Arduino IDE → 文件 → 示例 → Lib 选择 (BMC56M001) → 选择 (Star) → 选择范例 (Node)
2. 示例说明：

a. 创建对象 & 模块初始化及配置

```
#include  "BMC56M001.h"
BMC56M001      BMC56(5,4); // TX 引脚连接开发板 D5，RX 引脚连接开发板 D4
#define   KEY1_Pin (8)      // 将 D8 与 GND 通过按键连接，此按键为 KEY1
#define   KEY2_Pin (9)      // 将 D9 与 GND 通过按键连接，此按键为 KEY2
uint8_t   Message_ShortAddr;
bool Flag_Pairing,Flag_PairSuccess;
uint8_t TXDATA[16] = {0},RXDATA[32] = {0};
uint8_t DATA,STATUS,len;
/******* 函数声明 *******/
uint8_t Sys_KEY(void);           // 获取按键状态
void RFMessage_Process();        // 根据按键状态执行相关动作
void Handle_RFPacket_Process(); // 获取数据包 & 获取对方设备 ID
void setup()
{
  /***** 按键初始化 *****/
  pinMode(KEY1_Pin, INPUT_PULLUP);
  pinMode(KEY2_Pin, INPUT_PULLUP);
  Serial.begin(115200);                 // 配置串口监视器
  BMC56.begin(BR_38400);                // 初始化模块，配置通信速率
  BMC56.setDeviceRole(Node_of_Star); // 选择设备角色
}
```

b. 根据按键状态执行配对、发送配对包、发送数据包等操作，有接收到数据时获取数据并在串口监视器上显示

```
void loop()
{
  RFMessage_Process();        // 扫描按键
  Handle_RFPacket_Process(); // 扫描是否接收到数据
}
```

c. 获取按键状态函数

```
uint8_t Sys_KEY(void)
{
  if(!digitalRead(KEY1_Pin))
  {
    delay(60);
    if(!digitalRead(KEY1_Pin))
    {
      return 0x01;
    }
  }
  if(!digitalRead(KEY2_Pin))
  {
    delay(60);
    if(!digitalRead(KEY2_Pin))
    {
      return 0x02;
    }
  }
  return 0x00;
}
```

d. 根据按键状态执行相关程序

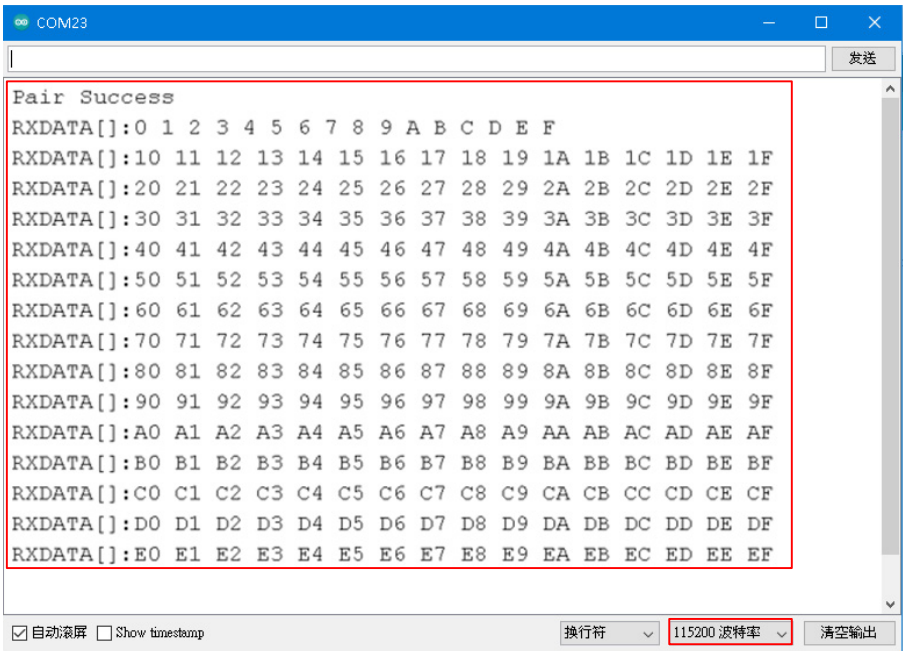
```
void RFMessage_Process()
{
  switch(Sys_KEY())
  {
    case 0x01:
    /****KEY1 按键有按下 ****/
    BMC56.setMode(Pairing_Mode); // 进入配对模式
    Flag_Pairing = TRUE;
    Flag_PairSuccess = FALSE;
    break;
    case 0x02:
    /****KEY2 按键有按下 ****/
    if(Flag_Pairing)              // 判断是否正在配对中
    {
      BMC56.writePairPackage();   // 发送配对请求包
    }
    if(Flag_PairSuccess)          // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
```

```
      BMC56.writeRFData(Message_ShortAddr,16,TXDATA); // 发送数据包
    }
    break;
  }
}
```

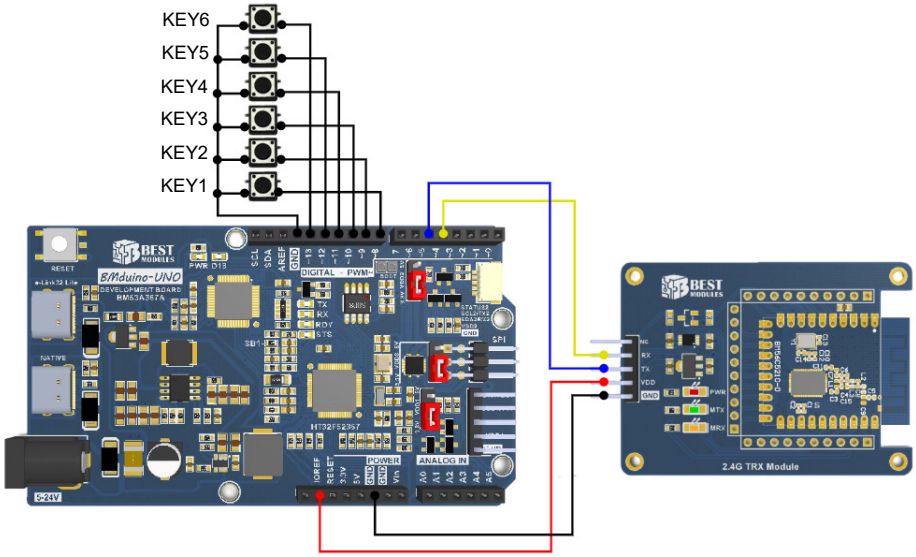
e. 接收到数据时在串口监视器上显示

```
void Handle_RFPacket_Process()
{
  if(Flag_Pairing)                          // 判断是否在配对中
  {
    STATUS = BMC56.getPairStatus();         // 获取配对情况
    if(STATUS == 1) // pairing success      // 判断是否配对成功
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = TRUE;
      Message_ShortAddr = BMC56.getShortAddress();// 获取短地址
      BMC56.setMode(Rx_Mode);               // 进入 RX 模式
      Serial.println("Pair Success");// 在串口监视器上显示 "Pair Success"
    }
    if(STATUS == 2)                         // 判断是否配对失败
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
    if(STATUS == 3)                         // 判断是否配对超时
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
  }
  if(Flag_PairSuccess)                      // 判断是否已经配对成功
  {
    if(BMC56.isInfoAvailable())             // 判断是否有数据待读取
    {
      STATUS = BMC56.readRFData(RXDATA,len);  // 读取数据
      if(STATUS == 1)                       // 判断读取的数据是否为数据包
      {
        Serial.print("RXDATA[]:");    // 串口监视器上显示读取的数据
        for(uint8_t temp=0;temp<len;temp++)
        {
          Serial.print(RXDATA[temp],HEX);
          Serial.print(" ");
        }
        Serial.println(" ");
      }
    }
  }
}
```

3. 打开串口监视器，波特率选择 115200；串口监视器显示如下



范例 3：Concentrator



实物连接示意图

范例实现功能：设置模块角色为 Concentrator 并且与 Node 角色的模块搭配形成 Star 网络拓扑并且进行配对以及数据交流。

此范例需搭配范例 Node 使用。

步骤如下：

(1). Node 端与 Concentrator 端配对

两个模块皆按下 KEY1 (D8)，进入配对模式：配对时间为 8 秒，此时 MRX 指示灯闪烁；Node 端短按 KEY2 (D9) 会发送配对请求封包，配对成功 MRX 指示灯熄灭，并且在 Node 端与 Concentrator 端的串口监视器上显示配对成功，若 8 秒内没有配对成功 MRX 指示灯也会熄灭，用户可再次重新配对。

(2). 接收与发送数据切换

Node 端与 Concentrator 端配对成功后，都进入接收模式，用户可短按任意 Node 端 KEY2 (D9) 来发送数据包给配对好的 Concentrator 端；或短按 Concentrator 端 KEY2~KEY6 (D9~D13) 来发送数据包给配对好之不同 Node。

(3). Node 端给 Concentrator 端发送数据

用户可短按 Node 端 KEY2 (D9) 来发送数据封包，此时 Node 端 MTX 指示灯会闪烁一次指示有数据包发出。Concentrator 端接收到数据时 MRX 指示灯会闪烁一次指示有接收到数据包，并在串口监视器上显示接收到的数据。

(4). Concentrator 端给 Node 端发送数据

用户可短按 Concentrator 端 KEY2 (D9) 来发送数据封包，此时 Concentrator 端 MTX 指示灯会闪烁一次，指示有数据包发出。Node 端接收到数据时 MRX 指示灯会闪烁一次指示有接收到数据包，并在串口监视器上显示接收到的数据。若有超过一个以上 Node，可按照配对顺序短按 Concentrator 端 KEY2~KEY6 (D9~D13) 分别对不同 Node 发送数据。

1. 范例打开方式： Arduino IDE → 文件 → 示例 → Lib 选择 (BMC56M001) → 选择 (Star) → 选择范例 (Concentrator)
2. 示例说明：

a. 创建对象 & 模块初始化及配置

```
#include  "BMC56M001.h"
BMC56M001     BMC56(5,4); // TX 引脚连接开发板 D5，RX 引脚连接开发板 D4
#define   KEY1_Pin (8)      // 将 D8 与 GND 通过按键连接，此按键为 KEY1
#define   KEY2_Pin (9)      // 将 D9 与 GND 通过按键连接，此按键为 KEY2
#define   KEY3_Pin (10)     // 将 D10 与 GND 通过按键连接，此按键为 KEY3
#define   KEY4_Pin (11)     // 将 D11 与 GND 通过按键连接，此按键为 KEY4
#define   KEY5_Pin (12)     // 将 D12 与 GND 通过按键连接，此按键为 KEY5
#define   KEY6_Pin (13)     // 将 D13 与 GND 通过按键连接，此按键为 KEY6
bool Flag_Pairing,Flag_PairSuccess;
uint8_t TXDATA[16] = {0},RXDATA[32] = {0};
uint8_t DATA,STATUS,len;
/******* 函数声明 *******/
uint8_t Sys_KEY(void);           // 获取按键状态
void RFMessage_Process();        // 根据按键状态执行相关动作
void Handle_RFPacket_Process(); // 获取数据包 & 获取对方设备 ID
void setup()
{
  /***** 按键初始化 *****/
  pinMode(KEY1_Pin, INPUT_PULLUP);
  pinMode(KEY2_Pin, INPUT_PULLUP);
  pinMode(KEY3_Pin, INPUT_PULLUP);
  pinMode(KEY4_Pin, INPUT_PULLUP);
```

```
  pinMode(KEY5_Pin, INPUT_PULLUP);
  pinMode(KEY6_Pin, INPUT_PULLUP);
  Serial.begin(115200);                          // 配置串口监视器
  BMC56.begin(BR_38400);                         // 初始化模块，配置通信速率
  BMC56.setDeviceRole(Concentrator_of_Star); // 选择设备角色
}
```

b. 根据按键状态执行配对、发送配对包、发送数据包等操作，有接收到数据时获取数据并在串口监视器上显示

```
void loop()
{
  RFMessage_Process();        // 扫描按键
  Handle_RFPacket_Process(); // 扫描是否接收到数据
}
```

c. 获取按键状态函数

```
uint8_t Sys_KEY(void)
{
  if(!digitalRead(KEY1_Pin))
  {
    delay(60);
    if(!digitalRead(KEY1_Pin))
    {return 0x01;}
  }
  if(!digitalRead(KEY2_Pin))
  {
    delay(60);
    if(!digitalRead(KEY2_Pin))
    {return 0x02;}
  }
  if(!digitalRead(KEY3_Pin))
  {
    delay(60);
    if(!digitalRead(KEY3_Pin))
    {return 0x03;}
  }
  if(!digitalRead(KEY4_Pin))
  {
    delay(60);
    if(!digitalRead(KEY4_Pin))
    {return 0x04;}
  }
  if(!digitalRead(KEY5_Pin))
  {
    delay(60);
    if(!digitalRead(KEY5_Pin))
    {return 0x05;}
  }
  if(!digitalRead(KEY6_Pin))
  {
    delay(60);
    if(!digitalRead(KEY6_Pin))
    {return 0x06;}
  }
  return 0x00;
}
```

d. 根据按键状态执行相关程序

```
void RFMessage_Process()
{
  switch(Sys_KEY())
  {
    case 0x01:
    /****KEY1 按键有按下 ****/
    if(BMC56.setMode(Pairing_Mode)) // 进入配对模式
    {
      Flag_Pairing = TRUE;
      Flag_PairSuccess = FALSE;
    }
    break;
    case 0x02:
    /****KEY2 按键有按下 ****/
    if(Flag_PairSuccess)        // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Node1_ShortAddr,16,TXDATA);// 向 Node1 发送数据包
    }
    break;
    case 0x03:
    /****KEY3 按键有按下 ****/
    if(Flag_PairSuccess)        // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Node2_ShortAddr,16,TXDATA);// 向 Node2 发送数据包
    }
    break;
    case 0x04:
    /****KEY4 按键有按下 ****/
    if(Flag_PairSuccess)        // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Node3_ShortAddr,16,TXDATA);// 向 Node3 发送数据包
    }
    break;
    case 0x05:
```

```
    /****KEY5 按键有按下 ****/
    if(Flag_PairSuccess)        // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Node4_ShortAddr,16,TXDATA);// 向 Node4 发送数据包
    }
    break;
    case 0x06:
    /****KEY6 按键有按下 ****/
    if(Flag_PairSuccess)        // 判断是否已配对成功
    {
      for(uint8_t temp=0;temp<16;temp++)
      {
        TXDATA[temp] = DATA++; // 发送的数据从 0x00~0x0f、0x10~0x1f……、
                               // 0xf0~0xff 循环
      }
      BMC56.writeRFData(Node5_ShortAddr,16,TXDATA);// 向 Node5 发送数据包
    }
    break;
  }
}
```

e. 接收到数据时在串口监视器上显示

```
void Handle_RFPacket_Process()
{
  if(Flag_Pairing)                    // 判断是否在配对中
  {
    STATUS = BMC56.getPairStatus();   // 获取配对情况
    if(STATUS == 1)                   // 判断是否配对成功
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = TRUE;
      BMC56.setMode(Rx_Mode);         // 进入 RX 模式
      Serial.println("Pair Success");// 在串口监视器上显示 "Pair Success"
    }
    if(STATUS == 2)                   // 判断是否配对失败
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
    if(STATUS == 3)                   // 判断是否配对超时
    {
      Flag_Pairing = FALSE;
      Flag_PairSuccess = FALSE;
    }
  }
```

```
  if(Flag_PairSuccess)                  // 判断是否已经配对成功
  {
    if(BMC56.isInfoAvailable())         // 判断是否有数据待读取
    {
      STATUS = BMC56.readRFData(RXDATA,len); // 读取数据
      if(STATUS == 1)                   // 判断读取的数据是否为数据包
      {
        Serial.print("RXDATA[]:");     // 串口监视器上显示读取的数据
        for(uint8_t temp=0;temp<len;temp++)
        {
          Serial.print(RXDATA[temp],HEX);
          Serial.print(" ");
        }
        Serial.println(" ");
      }
    }
  }
}
```

3. 打开串口监视器，波特率选择 115200；串口监视器显示如下