

# STM32 上移植 FreeModbus RTU

前段时间了解了一下 Modbus 协议，移植了一次，不成功  
<http://www.openedv.com/thread-67471-1-1.html>，不过找到方法之后，再次完善  
在官网上面下载源码，度娘上面有些被人家修改过的，喜欢原汁原味的朋友去官网上下载，  
本帖附件也附上源码。

开发平台：原子哥[探索者开发板](#) V2.2

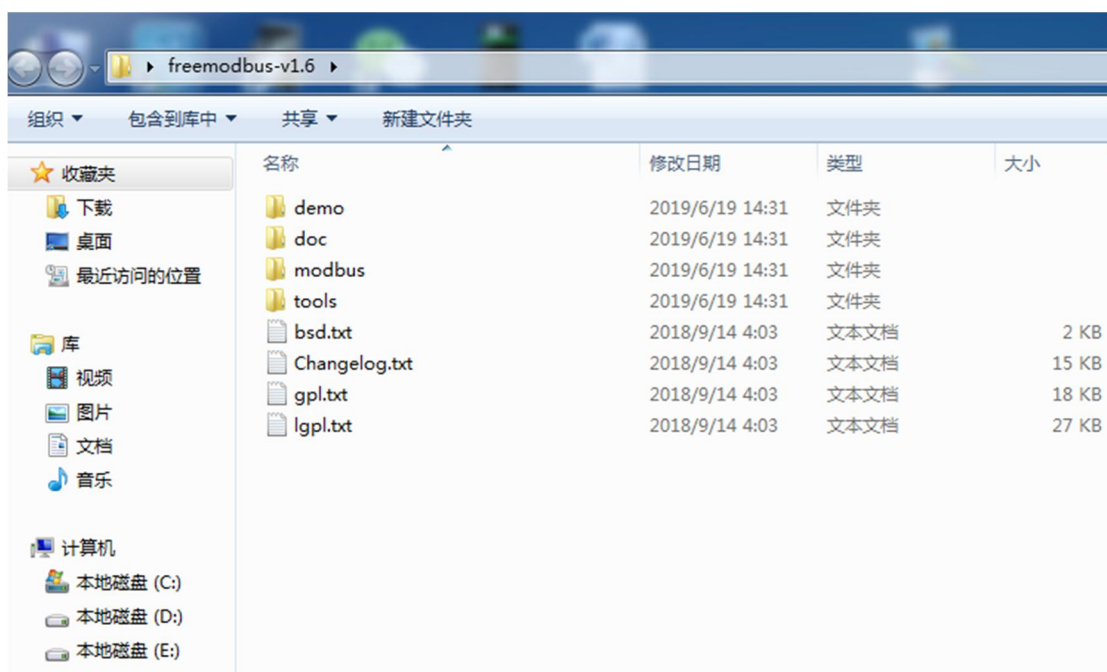
编译环境：MDK5.17

STM32 官方库函数：V1.5.1

FreeModbu 版本：V1.6.0

## 一、解压 freemodbus v1.6 源码

看到如下文件目录结构

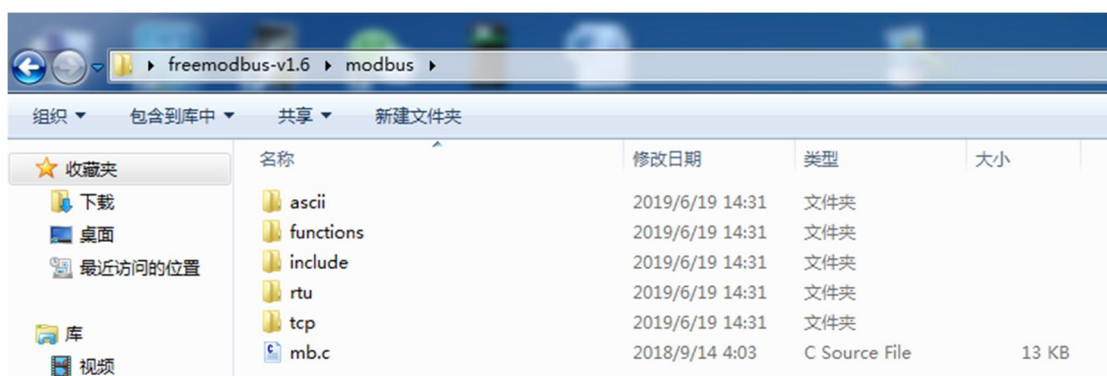


文件夹 demo 就是官方针对不同平台移植的测试代码

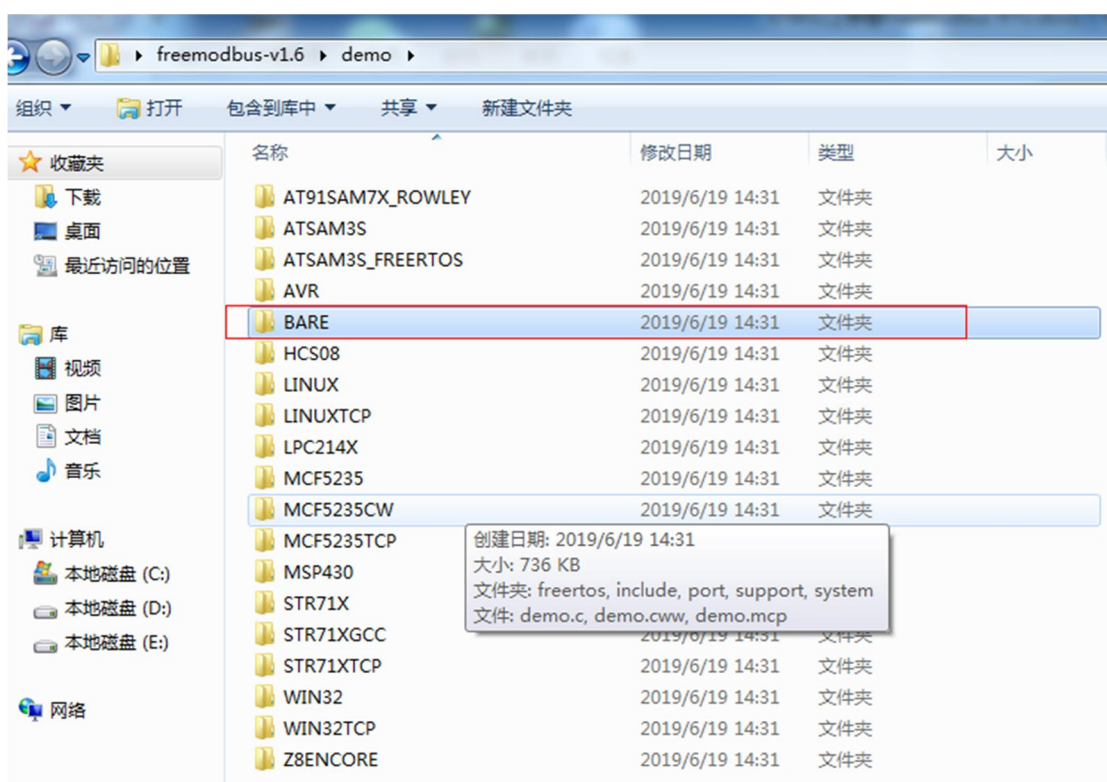
文件夹 doc 是一些说明性文档

文件夹 modbus 就是功能实现的源码所在了

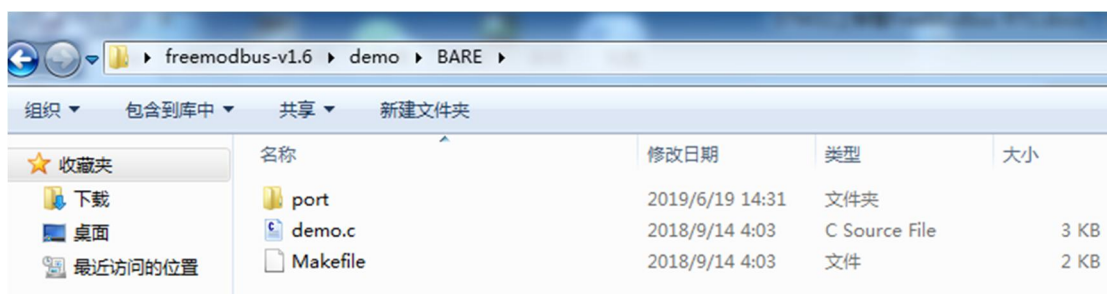
文件夹 tools 是上位机软件



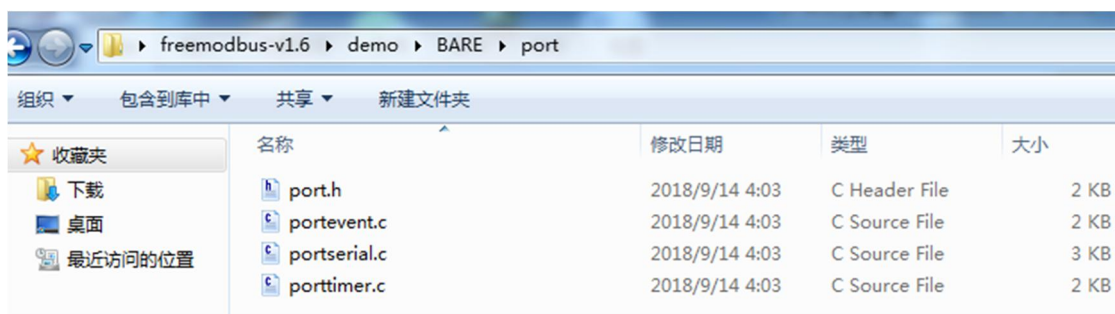
图一、FREEMODBUS V1.6 压缩后 MODBUS 文件夹内容



图二、FREEMODBUS V1.6 压缩后 DEMO 文件夹内容



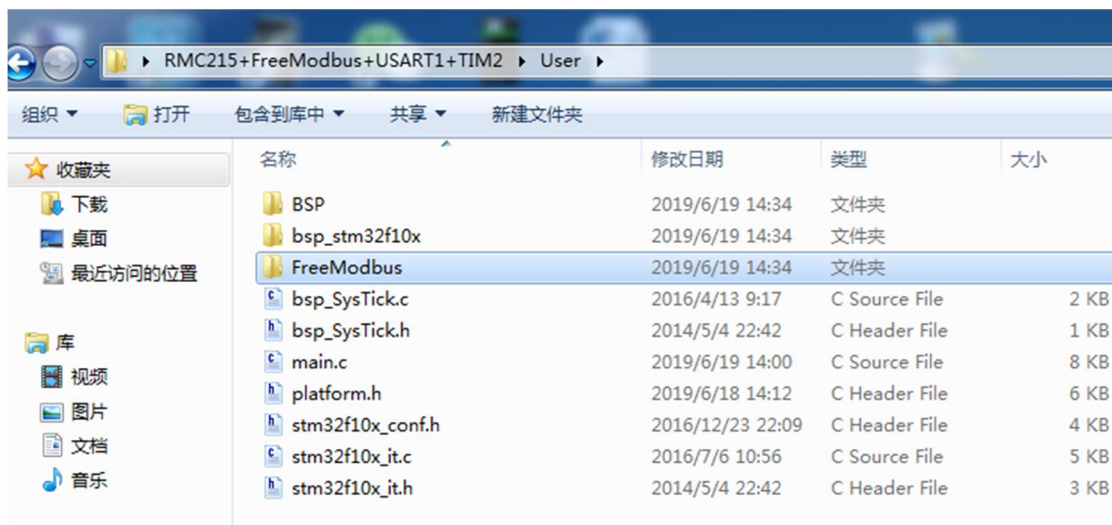
图三、FREEMODBUS V1.6 压缩后 DEMO\BARE 文件夹内容



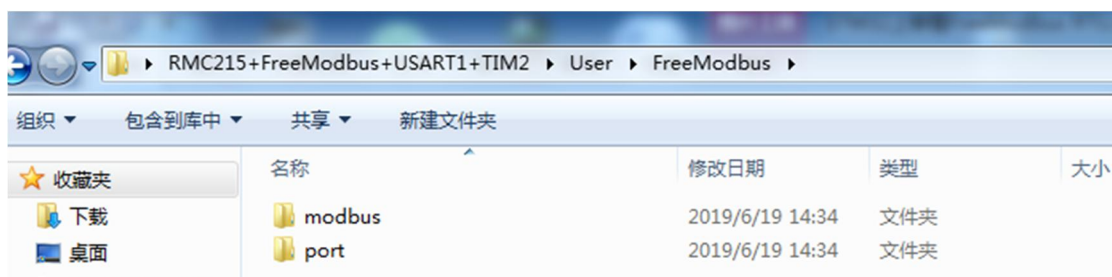
图四、FREEMODBUS V1.6 压缩后 DEMO\BARE\port 文件夹内容

## 二、建立工程

2.1、新建一个工程（这个就随意了），工程项目所有的文件夹名为 RMC215+FreeModbus+USART1+TIM2，在该文件夹内再建立一个文件夹 FreeModbus，然后在文件夹 FreeModbus 下再分别建立一个 modbus 文件夹和 port 文件夹，参见下图。

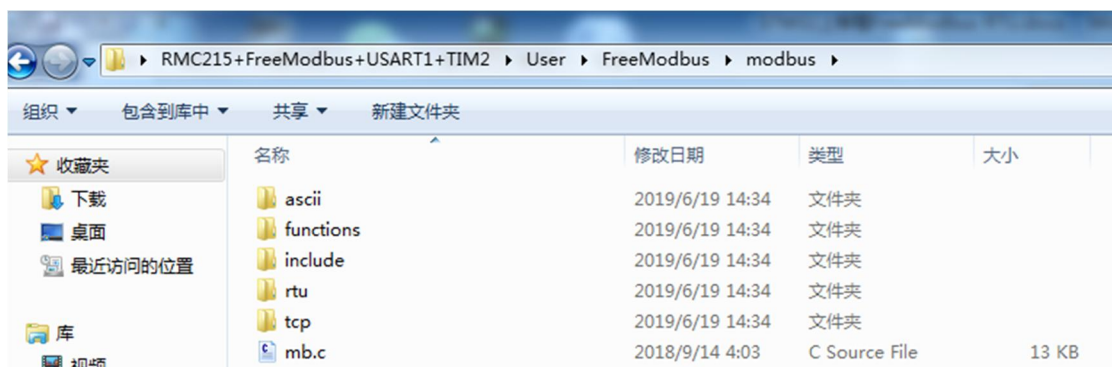


图五、工程项目所在文件夹下创建一个 FREEMODBUS 文件夹

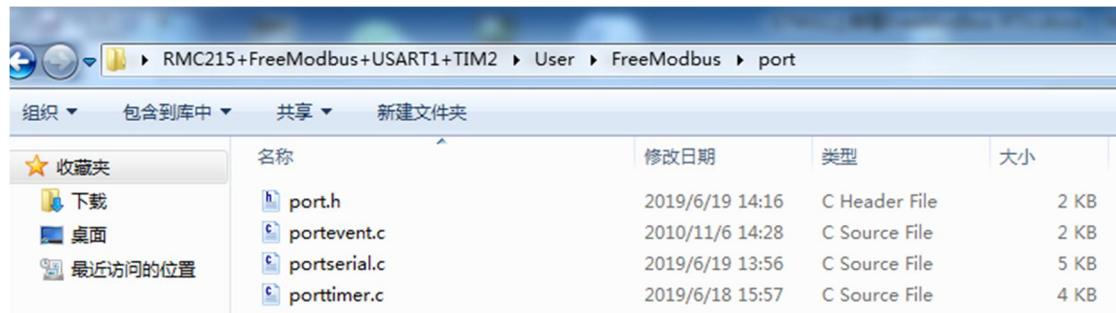


图六、工程项目所在文件夹 FREEMODBUS 下创建 modbus 文件夹和 port 文件夹

2.2、将 FreeModbusV1.6 下的 modbus 所有文件夹及文件拷贝到新建工程项目 FreeModbus 文件夹下的 modbus 子文件夹下。



2.3、进入 FreeModbusV1.6 下的 demo 文件夹，看到有各个平台的测试代码文件夹，没看到 STM32 的，但是看到 BARE 这个不带任何平台的代码文件，将 FreeModbusV1.6 下 \demo\BARE\port 下的所有文件拷贝到新建工程项目 FreeModbus 文件夹下\port 文件夹中。



其中：

- (1) port.h 需要修改。
- (2) portevent.c 不需要任何修改
- (3) portserial.c 需要修改
- (4) porttimer.c 需要修改。
- (5) 另外还需要在 main 函数增加 4 个回调函数。

(5.1) 操作输入寄存器的回调函数 eMBErrorCode eMBRegInputCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNRegs )

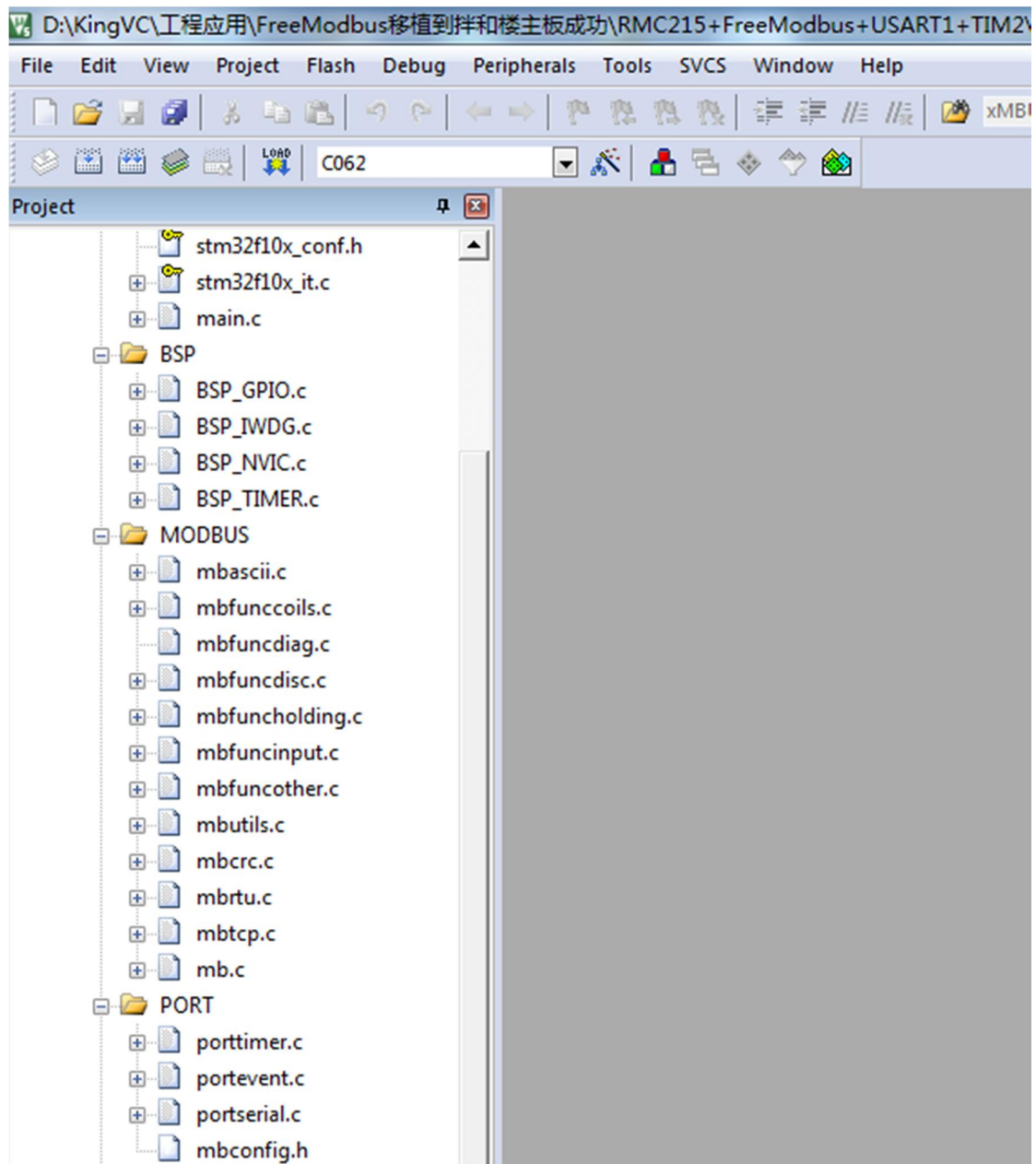
(5.2) 操作保持寄存器的回调函数 eMBErrorCode eMBRegHoldingCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNRegs, eMBRegisterMode eMode )

(5.3) 操作线圈的的回调函数 eMBErrorCode eMBRegCoilsCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNCoils, eMBRegisterMode eMode )

(5.4) 操作离散寄存器的的回调函数 eMBErrorCode eMBRegDiscreteCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNDiscrete )

2.3、打开 MDK（用 IAR 也行，随意了），建立工程。。（省略 1 万字）

下图是我创建好的移植工程项目。



### 三、添加代码

3.1、打开 portserial.c 文件，这个是移植串口的，不管是 ASCII 模式还是 RTU 模式都需要串口支持的，void vMBPortSerialEnable( BOOL xRxEnable, BOOL xTxEnable )函数，使能或失能串口的，移植代码如下：

```
void
vMBPortSerialEnable( BOOL xRxEnable, BOOL xTxEnable )
{
    /* If xRXEnable enable serial receive interrupts. If xTxENable enable
```

```

* transmitter empty interrupts.
*/

if (xRxEnable) //接收使能
{
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE); //使能接收中断
    GPIO_ResetBits(GPIOG, GPIO_Pin_8); //设置 RS485 接收
}
else //失能
{
    USART_ITConfig(USART2, USART_IT_RXNE, DISABLE); //失能接收中断
    GPIO_SetBits(GPIOG, GPIO_Pin_8); //设置 RS485 发送
}

if (xTxEnable) //发送使能
{
    USART_ITConfig(USART2, USART_IT_TC, ENABLE); //使能
    GPIO_SetBits(GPIOG, GPIO_Pin_8); //设置 RS485 发送
}
else //失能
{
    USART_ITConfig(USART2, USART_IT_TC, DISABLE); //失能
    GPIO_ResetBits(GPIOG, GPIO_Pin_8); //设置 RS485 接收
}
}

```

3.1.1、串口初始化函数 `BOOL xMBPortSerialInit( UCHAR ucPORT, ULONG ulBaudRate, UCHAR ucDataBits, eMBParity eParity )` , 使用的是串口 2 进行通讯

```

BOOL
xMBPortSerialInit( UCHAR ucPORT, ULONG ulBaudRate, UCHAR ucDataBits, eMBParity eParity )
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef  NVIC_InitStructure;

    (void)ucPORT; //不修改串口号
    (void)ucDataBits; //不修改数据位长度
    (void)eParity; //不修改检验格式

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOG, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    //
    //管脚复用

```



```

//
GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);
//
//发送管脚 PA.02
//接收管脚 PA.03
//
GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_2 | GPIO_Pin_3;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
//
//485 芯片发送接收控制管脚
//
GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_8;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
GPIO_Init(GPIOG, &GPIO_InitStructure);
//
//配置串口参数
//
USART_InitStructure.USART_BaudRate      = ulBaudRate; //只修改波特率
USART_InitStructure.USART_WordLength    = USART_WordLength_8b;
USART_InitStructure.USART_StopBits      = USART_StopBits_1;
USART_InitStructure.USART_Parity        = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Parity        = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART2, &USART_InitStructure);
//
//使能串口
//
USART_Cmd(USART2, ENABLE);
//
//配置中断优先级
//
NVIC_InitStructure.NVIC_IRQChannel      = USART2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority      = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd            = ENABLE;
NVIC_Init(&NVIC_InitStructure);

```



```

    return TRUE;
}

```

### 3.1.2、发送一个字节函数 BOOL xMBPortSerialPutByte( CHAR ucByte )

```

BOOL
xMBPortSerialPutByte( CHAR ucByte )
{
    /* Put a byte in the UARTs transmit buffer. This function is called
     * by the protocol stack if pxMBFrameCBTransmitterEmpty( ) has been
     * called. */

    USART_SendData(USART2, ucByte); //发送一个字节

    return TRUE;
}

```

### 3.1.3、接收一个字节函数 BOOL xMBPortSerialGetByte( CHAR \* pucByte )

```

BOOL
xMBPortSerialGetByte( CHAR * pucByte )
{
    /* Return the byte in the UARTs receive buffer. This function is called
     * by the protocol stack after pxMBFrameCBByteReceived( ) has been called.
     */

    *pucByte = USART_ReceiveData(USART2); //接收一个字节

    return TRUE;
}

```

### 3.1.4、串口中断服务函数 void USART2\_IRQHandler(void)

```

/**
 *
 * @Name : 串口 2 中断服务函数
 *
 */
void USART2_IRQHandler(void)
{
    if (USART_GetITStatus(USART2, USART_IT_RXNE) == SET) //接收中断
    {
        prvvUARTRxISR();
    }
}

```

```

        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
    }

    if (USART_GetITStatus(USART2, USART_IT_TC) == SET) //发送中断
    {
        prvUARTTxReadyISR();
        USART_ClearITPendingBit(USART2, USART_IT_TC);
    }
}

```

### 3.2、打开 porttimer.c 文件，RTU 模式需要定时器支持，定时器初始化函数 BOOL

```

xMBPortTimersInit( USHORT usTim1Timerout50us )
BOOL
xMBPortTimersInit( USHORT usTim1Timerout50us )
{
    // return FALSE;

    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    NVIC_InitTypeDef      NVIC_InitStructure;

    uint16_t PrescalerValue = 0;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    //
    //HCLK 为 72MHz
    //时基频率 72 / ( 1 + Prescaler) = 20KHz
    //
    PrescalerValue = (uint16_t)((SystemCoreClock / 20000) - 1);
    //
    //初始化定时器参数
    //
    TIM_TimeBaseStructure.TIM_Period = (uint16_t)usTim1Timerout50us;
    TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    //
    //使能预装
    //
    TIM_ARRPreloadConfig(TIM2, ENABLE);

    //
    //初始化中断优先级

```

```
//
NVIC_InitStructure.NVIC_IRQChannel      = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority    = 2;
NVIC_InitStructure.NVIC_IRQChannelCmd          = ENABLE;
NVIC_Init(&NVIC_InitStructure);

TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
TIM_Cmd(TIM2, DISABLE);

return TRUE;
}
```

### 3.2.1、定时器使能函数 void vMBPortTimersEnable( )

```
void
vMBPortTimersEnable( )
{
    /* Enable the timer with the timeout passed to xMBPortTimersInit( ) */

    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    TIM_SetCounter(TIM2, 0x00000000);
    TIM_Cmd(TIM2, ENABLE);
}
```

### 3.2.2、定时器失能函数 void vMBPortTimersDisable( )

```
void
vMBPortTimersDisable( )
{
    /* Disable any pending timers. */

    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
    TIM_SetCounter(TIM2, 0x00000000);
    TIM_Cmd(TIM2, DISABLE);
}
```

### 3.2.3、定时器中断服务函数 void TIM2\_IRQHandler(void)

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

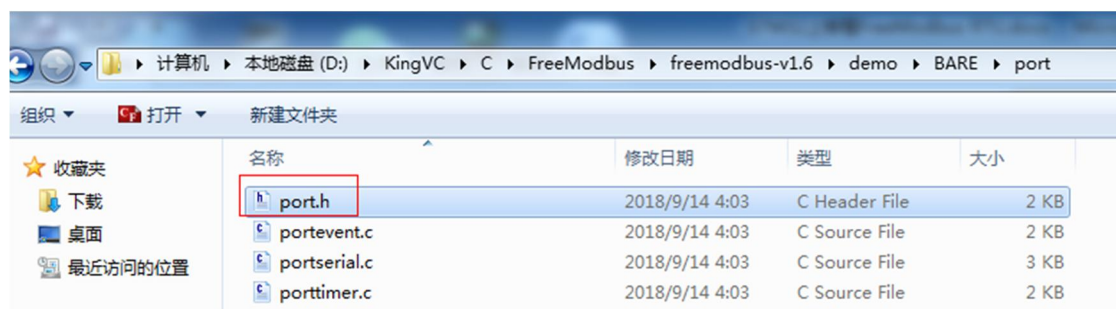
```

* @Name : 定时器 4 中断服务函数
*****

**/
void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        prvTIMERExpiredISR();
    }
}

```

四、打开 port.h 文件，这个文件可以从\demo\BARE 文件夹下拷贝过来的



4.1、这是\demo\BARE 文件夹下原 port.h 文件的内容

```

#ifndef _PORT_H
#define _PORT_H

#include <assert.h>
#include <inttypes.h>

#define INLINE inline
#define PR_BEGIN_EXTERN_C extern "C" {
#define PR_END_EXTERN_C }

#define ENTER_CRITICAL_SECTION( )
#define EXIT_CRITICAL_SECTION( )

typedef uint8_t BOOL;

typedef unsigned char UCHAR;
typedef char CHAR;

typedef uint16_t USHORT;
typedef int16_t SHORT;

typedef uint32_t ULONG;
typedef int32_t LONG;

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#endif

```

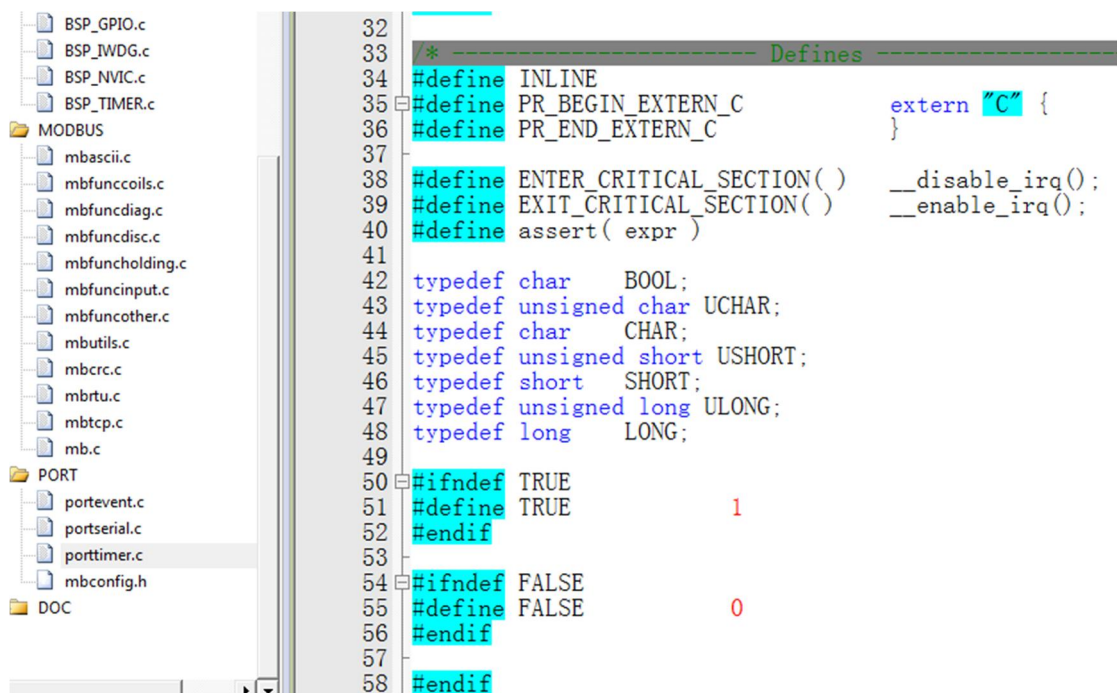
在 port.h 文件补充完成以下两个函数 ENTER\_CRITICAL\_SECTION( ) 和 EXIT\_CRITICAL\_SECTION( )，这两个函数跟中断有关，开或者关中断。

```
33 /* ----- Defines ----- */
34 #define INLINE
35 #define PR_BEGIN_EXTERN_C      extern "C" {
36 #define PR_END_EXTERN_C      }
37
38 #define ENTER_CRITICAL_SECTION( )    __disable_irq();
39 #define EXIT_CRITICAL_SECTION( )    __enable_irq();
40 #define assert( expr )
41
```

在 port.h 文件中增加断言宏定义

```
#define assert( expr )
```

4.2、 以下是工程项目文件夹下修改后的 port.h 文件内容



```
32
33 /* ----- Defines ----- */
34 #define INLINE
35 #define PR_BEGIN_EXTERN_C      extern "C" {
36 #define PR_END_EXTERN_C      }
37
38 #define ENTER_CRITICAL_SECTION( )    __disable_irq();
39 #define EXIT_CRITICAL_SECTION( )    __enable_irq();
40 #define assert( expr )
41
42 typedef char      BOOL;
43 typedef unsigned char UCHAR;
44 typedef char      CHAR;
45 typedef unsigned short USHORT;
46 typedef short     SHORT;
47 typedef unsigned long  ULONG;
48 typedef long      LONG;
49
50 #ifndef TRUE
51 #define TRUE      1
52 #endif
53
54 #ifndef FALSE
55 #define FALSE     0
56 #endif
57
58 #endif
```

4.3、 分别完善以下 4 个回调函数

4.3.1、 操作输入寄存器 eMBCErrorcode eMBRegInputCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNRegs )

4.3.2、 操作保持寄存器 eMBCErrorcode eMBRegHoldingCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNRegs, eMBRegisterMode eMode )

4.3.3、 操作线圈 eMBCErrorcode eMBRegCoilsCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNCoils, eMBRegisterMode eMode )

4.3.4、操作离散寄存器 eMBCErrorcode eMBRegDiscreteCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNDiscrete )

## 五、编写 main 函数

```
int main( void )
{
    //
    //初始化硬件
    //
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    eMBInit(MB_RTU, 0x0A, 1, 38400, MB_PAR_NONE); //初始化 FreeModbus
    eMBEnable(); //启动 FreeModbus

    while (1)
    {
        (void)eMBPoll(); //查询数据帧
        usRegInputBuf[0]++; //测试读取输入寄存器用
        if (usRegInputBuf[0] > 250) //防止超标
        {
            usRegInputBuf[0] = 0;
        }
    }
}
```

## 六、编译

问题来了，提示错误

```
*** Using Compiler 'V5.06 update 1 (build 61)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'FreeModbus_M4'
compiling main.c...
linking...
Error: L6915E: Library reports error: use no semihosting was requested, but _ttywrch was referenced
Finished: 0 information, 0 warning and 1 error messages.
".\Objects\FreeModbus_M4.axf" - 1 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:03
```

FreeModbus 作者使用了 assert 这货，各种百度，说什么乱七八糟的，没解决什么问题，最后还是自己解决了 <http://www.openedv.com/thread-67471-1-1.html>

在 main 函数最后增加代码

```
#ifndef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
```

```

*/
void assert_failed(uint8_t* file, uint32_t line)
{
    /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

    /* Infinite loop */
    while (1)
    {
    }
}
#else
void __aeabi_assert(const char * x1, const char * x2, int x3)
{
}
#endif

```

再次编译,发现没错误了,下载到板子上面运行,用串口助手( Modbus 调试精灵也行)发送命令 0A 04 00 01 00 01 61 71 读取输入寄存器,发现没返回数据,各种仿真之后,发现串口中断的问题,前面初始化使用了 TC,发送完成中断,打开手册看到了 TXE 这个,修改编译下载,再次发送命令,这回有返回了,不过数据不对,返回 0。

在 eMBErrorCode eMBRegInputCB( UCHAR \* pucRegBuffer, USHORT usAddress, USHORT usNRegs )函数中将 usAddress 变量显示到液晶上面一看,不对,比发送的地址多了 1,跳转到函数 eMBFuncReadInputRegister( UCHAR \* pucFrame, USHORT \* usLen )一看,解释之后来了一句 usRegAddress++;,赶紧注释掉,再来,这回对了。**(后记：实际上注释掉这段代码是不对的, FREEMODBUS 原作者是完全按照 MODBUS 协议在编程,数据不对不应该在这里屏蔽原作者的代码,而是应该在回调函数中去作文章)**

```

56
57 eMBException
58 eMBFuncReadInputRegister( UCHAR * pucFrame, USHORT * usLen )
59 {
60     USHORT      usRegAddress;
61     USHORT      usRegCount;
62     UCHAR       *pucFrameCur;
63
64     eMBException  eStatus = MB_EX_NONE;
65     eMBErrorCode  eRegStatus;
66
67     if( *usLen == ( MB_PDU_FUNC_READ_SIZE + MB_PDU_SIZE_MIN ) )
68     {
69         usRegAddress = ( USHORT ) ( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF] << 8 );
70         usRegAddress |= ( USHORT ) ( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF + 1] );
71         usRegAddress++;
72
73         usRegCount = ( USHORT ) ( pucFrame[MB_PDU_FUNC_READ_REGCNT_OFF] << 8 );
74         usRegCount |= ( USHORT ) ( pucFrame[MB_PDU_FUNC_READ_REGCNT_OFF + 1] );
75

```



接着测试其他的，也发现了最后操作的地址比发送过去的地址多了 1，又打开相关的函数文件，都是在地址解释之后有 usRegAddress++;这句存在，赶紧注释掉，再来，这回正确了。（后记：实际上注释掉这段代码是不对的，FREEMODBUS 原作者是完全按照 MODBUS 协议在编程，数据不对不应该在这里屏蔽原作者的代码，而是应该在回调函数中去作文章）。

```
73 #if MB_FUNC_WRITE_HOLDING_ENABLED > 0
74
75 eMBException
76 eMBFuncWriteHoldingRegister( UCHAR * pucFrame, USHORT * usLen )
77 {
78     USHORT          usRegAddress;
79     eMBException     eStatus = MB_EX_NONE;
80     eMBCode          eRegStatus;
81
82     if( *usLen == ( MB_PDU_FUNC_WRITE_SIZE + MB_PDU_SIZE_MIN ) )
83     {
84         usRegAddress = ( USHORT )( pucFrame[MB_PDU_FUNC_WRITE_ADDR_OFF] << 8 );
85         usRegAddress |= ( USHORT )( pucFrame[MB_PDU_FUNC_WRITE_ADDR_OFF + 1] );
86         usRegAddress++;
87
88         /* Make callback to update the value. */
89         eRegStatus = eMBRegHoldingCB( &pucFrame[MB_PDU_FUNC_WRITE_VALUE_OFF],
90                                     usRegAddress, 1, MB_REG_WRITE );
91     }
92
93     eMBException
94     eMBFuncReadDiscreteInputs( UCHAR * pucFrame, USHORT * usLen )
95     {
96         USHORT          usRegAddress;
97         USHORT          usDiscreteCnt;
98         UCHAR          ucNBytes;
99         UCHAR          *pucFrameCur;
100
101         eMBException     eStatus = MB_EX_NONE;
102         eMBCode          eRegStatus;
103
104         if( *usLen == ( MB_PDU_FUNC_READ_SIZE + MB_PDU_SIZE_MIN ) )
105         {
106             usRegAddress = ( USHORT )( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF] << 8 );
107             usRegAddress |= ( USHORT )( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF + 1] );
108             usRegAddress++;
109
110             usDiscreteCnt = ( USHORT )( pucFrame[MB_PDU_FUNC_READ_DISCCNT_OFF] << 8 );
111             usDiscreteCnt |= ( USHORT )( pucFrame[MB_PDU_FUNC_READ_DISCCNT_OFF + 1] );
112         }
113
114         eMBException
115         eMBFuncReadCoils( UCHAR * pucFrame, USHORT * usLen )
116         {
117             USHORT          usRegAddress;
118             USHORT          usCoilCount;
119             UCHAR          ucNBytes;
120             UCHAR          *pucFrameCur;
121
122             eMBException     eStatus = MB_EX_NONE;
123             eMBCode          eRegStatus;
124
125             if( *usLen == ( MB_PDU_FUNC_READ_SIZE + MB_PDU_SIZE_MIN ) )
126             {
127                 usRegAddress = ( USHORT )( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF] << 8 );
128                 usRegAddress |= ( USHORT )( pucFrame[MB_PDU_FUNC_READ_ADDR_OFF + 1] );
129                 usRegAddress++;
130
131                 usCoilCount = ( USHORT )( pucFrame[MB_PDU_FUNC_READ_COILCNT_OFF] << 8 );
132                 usCoilCount |= ( USHORT )( pucFrame[MB_PDU_FUNC_READ_COILCNT_OFF + 1] );
133             }
134         }
135     }
136 }
```

## 七、回到刚才的串口发送中断标志 TC 和 TXE 的问题上来，对照手册一看

### 位 7 TXE：发送数据寄存器为空 (Transmit data register empty)

当 TDR 寄存器的内容已传输到移位寄存器时，该位由硬件置 1。如果 USART\_CR1 寄存器中 TXEIE 位 = 1，则会生成中断。通过对 USART\_DR 寄存器执行写入操作将该位清零。

0：数据未传输到移位寄存器

1：数据传输到移位寄存器

注意：单缓冲区发送期间使用该位。

### 位 6 TC：发送完成 (Transmission complete)

如果已完成对包含数据的帧的发送并且 TXE 置 1，则该位由硬件置 1。如果 USART\_CR1 寄存器中 TCIE = 1，则会生成中断。该位由软件序列清零（读取 USART\_SR 寄存器，然后写入 USART\_DR 寄存器）。TC 位也可以通过向该位写入 '0' 来清零。建议仅在多缓冲区通信时使用此清零序列。

0：传送未完成

1：传送已完成

大家对比一下说明，我就不解释了。

打开 mbriu.c 文件，在下面位置增加一段代码，修正 TC 中断标志发送的错误

```
186 eMErrorCode
187 eMBRTUSend( UCHAR ucSlaveAddress, const UCHAR * pucFrame, USHORT usLength )
188 {
189     eMErrorCode    eStatus = MB_ENOERR;
190     USHORT         usCRC16;
191
192     ENTER_CRITICAL_SECTION( );
193
194     /* Check if the receiver is still in idle state. If not we where to
195      * slow with processing the received frame and the master sent another
196      * frame on the network. We have to abort sending the frame.
197      */
198     if( eRcvState == STATE_RX_IDLE )
199     {
200         /* First byte before the Modbus-PDU is the slave address. */
201         pucSndBufferCur = ( UCHAR * ) pucFrame - 1;
202         usSndBufferCount = 1;
203
204         /* Now copy the Modbus-PDU into the Modbus-Serial-Line-PDU. */
205         pucSndBufferCur[MB_SER_PDU_ADDR_OFF] = ucSlaveAddress;
206         usSndBufferCount += usLength;
207
208         /* Calculate CRC16 checksum for Modbus-Serial-Line-PDU. */
209         usCRC16 = usMBCRC16( ( UCHAR * ) pucSndBufferCur, usSndBufferCount );
210         ucRTUBuf[usSndBufferCount++] = ( UCHAR )( usCRC16 & 0xFF );
211         ucRTUBuf[usSndBufferCount++] = ( UCHAR )( usCRC16 >> 8 );
212
213         /* Activate the transmitter. */
214         eSndState = STATE_TX_XMIT;
215
216         /*
217          * 插入以下代码完成一次发送，启动发送完成中断
218          */
219         xMBPortSerialPutByte( ( CHAR ) *pucSndBufferCur );
220         pucSndBufferCur++;
221         usSndBufferCount--;
222         /*
223          * 结束
224          */
225
226         vMBPortSerialEnable( FALSE, TRUE );
227     }
228     else
229     {
230         eStatus = MB_EIO;
231     }
232     EXIT_CRITICAL_SECTION( );
233 }
```

如果大家不想修改源码的话就用 TXE 这个标志，不过增加了以上代码再使用 TXE 标志

也是没有问题的，无所谓了。

好了到此为止，移植算是完成了，继续深究的朋友可以接着，否则就直接用吧。

最后，如果要移植到 RTU 模式下，请打开\FreeModbus\modbus\include\mbconfig.h

请配置：#define MB\_RTU\_ENABLED ( 1 )

请配置#define MB\_ASCII\_ENABLED ( 0 )

请配置#define MB\_TCP\_ENABLED ( 0 )

