本移植例程在11、15、16基础上完成，首先了解《11.FreeModbus模拟智能表》Modbus RTU的数据传输流程，其次掌握《15.网络客户端》《16.网络服务器》中RTT下基于LWIP的网络客户端数据的读写和网络服务器的端口监听流程，在此基础上完成Modbus TCP从机（服务器）的系统开发。

# 1．Modbus TCP基础

## 1.1 数据格式

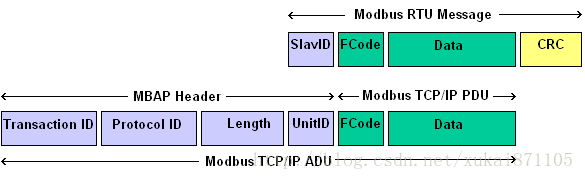
几个定义：

Modbus TCP主机在TCP/IP协议的网络中，为客户端。

Modbus TCP从机在TCP/IP协议的网络中，为服务器。

本例实现Modbus TCP从机，也就是在STM32F107的系统中，采用服务器模式监听502端口，当检测到有连接来到，接收数据并解析，当解析成功后，将从机上的保持寄存器或者输入寄存器数据通过TCP/IP协议发送出去。

modbus TCP和modbus RTU的区别可使用下图概括：



modbus TCP可以理解为发生在TCP上的应用层协议。其数据格式如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 区域 | 长度 | 描述 | 客户端 | 服务器 | 举例 |
| 传输标志 | 2  字节 | MODBUS 请求和响应传输过程中序列号 | 客户端生成 | 应答时复制该值 | 01 01 |
| 协议标志 | 2  字节 | Modbus协议默认为0 | 客户端生成 | 应答时复制该值 | 00 00 |
| 长度 | 2  字节 | 剩余部分的长度 | 客户端生成 | 应答时由服务器端生成 | 00 06  00 07 |
| 单元标志 | 1  字节 | 从机标志（从机地址） | 客户端生成 | 应答时复制该值 | 01 |
| 功能码 | 1  字节 | 03 功能码 | 客户端生成 | 应答时复制该值 | 03 |
| 主机发出的数据 | n  字节 | 从机标志（从机地址）+ 要读取数据的长度 | 客户端生成 |  | 00 00 00 02 |
| 从机发出的数据 | n字节 | 读取数据的长度 + 读取的数据值 | 服务器生成 | 应答时由服务器端生成 | 04 00 00 00 04 |

网络连接通后，主机发：01 01 00 00 00 06 01 03 00 00 00 02

从机回：01 01 00 00 00 07 01 03 04 00 00 00 04

01 01 为传输标志，从机应答时复制该值

00 00为协议标志，从机应答时复制该值

00 06为长度值，主机请求时为该字节后面所有的数据（01 03 00 00 00 02）长度

00 07为长度值，从机应答时为该字节后面所有的数据（01 03 04 00 00 00 04）长度

上表中阴影部分与Modbus RTU部分一致（仅缺少CRC校验部分）

01 03 00 00 00 02 同Modbus RTU协议中的主机命令去掉后面的CRC校验

01 03 04 00 00 00 04同Modbus RTU协议中的从机应答去掉后面的CRC校验

## 1.2 Modbus数据收发流程

首先说明一下 ModbusRTU模式的收发流程：

1. eMBInit 初始化模式、地址、串口、速率、校验。

2. eMBEnable 激活Modbus从机协议

3. eMBPoll 开始Modbus轮询。

在eMBPoll中，当有串口数据到来，串口接收中断调用prvvUARTRxISR(), 进而调用在eMBInit中指定的回调函数xMBRTUReceiveFSM处理接收数据……最后通过函数发送xMBPortEventPost( EV\_FRAME\_RECEIVED )向eMBPoll中发送事件。eMBPoll接收到该有数据到达事件后验证通过后，发送xMBPortEventPost( EV\_EXECUTE )。 eMBPoll接收到该执行事件后验证通过后，发送peMBFrameSendCur( ucMBAddress, ucMBFrame, usLength )， 将ModbusRTU报文发送出去。

如果是RTU模式，必须仔细编写以下函数。

case MB\_RTU:

pvMBFrameStartCur = eMBRTUStart;

pvMBFrameStopCur = eMBRTUStop;

peMBFrameSendCur = eMBRTUSend;

//报文接收函数

peMBFrameReceiveCur = eMBRTUReceive;

pvMBFrameCloseCur = MB\_PORT\_HAS\_CLOSE ? vMBPortClose : NULL;

//接收状态机

pxMBFrameCBByteReceived = xMBRTUReceiveFSM;

//发送状态机

pxMBFrameCBTransmitterEmpty = xMBRTUTransmitFSM;

//报文到达间隔检查

pxMBPortCBTimerExpired = xMBRTUTimerT35Expired;

//初始化RTU

eStatus = eMBRTUInit( ucMBAddress, ucPort, ulBaudRate, eParity );

如果是TCP模式：

pvMBFrameStartCur = eMBTCPStart;

pvMBFrameStopCur = eMBTCPStop;

peMBFrameReceiveCur = eMBTCPReceive;

peMBFrameSendCur = eMBTCPSend;

pvMBFrameCloseCur = MB\_PORT\_HAS\_CLOSE ? vMBTCPPortClose : NULL;

ucMBAddress = MB\_TCP\_PSEUDO\_ADDRESS;

eMBCurrentMode = MB\_TCP;

eMBState = STATE\_DISABLED;

# 2. FreeModbus TCP 的移植

经过整理后编程思路：

1. 在初始化过程中，如有数据到来，则发送xMBPortEventPost( EV\_FRAME\_RECEIVED ) 事件。

2. 后面由FreeModbus处理，最后在eMBTCPSend () 函数中，调用xMBTCPPortSendResponse()将整理好的数据通过网络发送出去。

在移植过程中，主要进行1）开始的初始化，2）接收数据在函数xMBTCPPortGetRequest中搬迁入FreeModbus,3）要发送的数据在xMBTCPPortSendResponse中从FreeModbus搬迁出来，并从TCP/IP中发送出去。

下面进行移植的具体操作：

## 2.1 修改mbconfig.h，使能TCP，关闭RTU和ASII

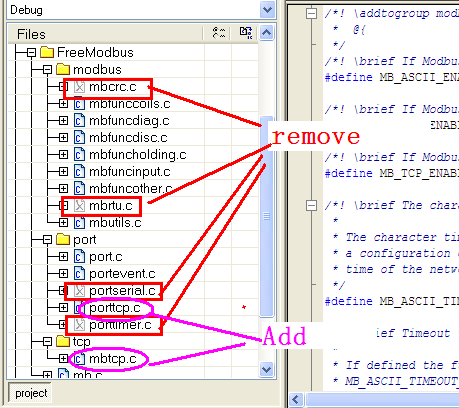
/\*! \brief If Modbus RTU support is enabled. \*/

#define MB\_RTU\_ENABLED ( 0 )

/\*! \brief If Modbus TCP support is enabled. \*/

#define MB\_TCP\_ENABLED ( 1 )

## 2.2 工程文件的添加和删除



添加：mbtcp.c porttcp.c

去除：mbcrc.c mbrtu.c portserial.c porttimer.c等4个文件

头文件也进行相应的调整。其中，porttcp.c是进行移植需要自行添加的最主要的文件，原协议中没有该文件。

## 2.3 porttcp.c编写详细说明

本文件参考<http://blog.csdn.net/xukai871105/article/details/21652287中的基于uIP的porttcp.c>的代码。

### 1）函数说明

本文件中，供FreeModbus调用的应用层函数一共有5个：

BOOL MBTCPPortInit( USHORT usTCPPort )—TCP初始化，监视502端口，与进入的连接建立通讯，并随时检测连接状态，如果连接断开，则要重新监视。

void vMBTCPPortClose( ) 为空

void MBTCPPortDisable( void ) 为空

BOOL MBTCPPortGetRequest( UCHAR \*\* ppucMBTCPFrame, USHORT \* usTCPLength )-- 接收数据搬迁入FreeModbus，\*ppucMBTCPFrame = &ucTCPRequestFrame[0];

BOOL MBTCPPortSendResponse( const UCHAR \* pucMBTCPFrame, USHORT usTCPLength )--要发送的数据从FreeModbus搬迁出来memcpy( ucTCPResponseFrame , pucMBTCPFrame , usTCPLength);，向nw\_thread线程发送事件， rt\_event\_send(&nw\_event, NW\_TX)， 将数据从LWIP中发送出去。

其余的全部是网络实现的函数，采用的编程思路与《15.网络客户端》《16.网络服务器》类似。

### 2）网络接收数据回调

MBTCPPortInit建立一个新连接后，开启网络数据接收线程nw\_thread，当接收到数据，自动调用网络数据接收回调函数rx\_callback,该函数向nw\_event发送一个接收事件，在nw\_thread进行接收。

### 3）向FreeModbus的modbus poll发送消息

nw\_thread里的接收事件调用process\_rx\_data函数处理接收到的数据，并向 modbus poll发送消息xMBPortEventPost( EV\_FRAME\_RECEIVED )。

### 4）LWIP网络数据的发送

Nw\_thread接收到事件NW\_TX后，向LWIP发送Modbus应答数据包

netconn\_write(newconn, ucTCPResponseFrame , ucTCPResponseLen, NETCONN\_NOCOPY );

# 3.调试结果

打开工程，在finsh shell中运行：test\_modbus()，开启网络502端口监听。

打开TCP/UDP调试软件，创建连接，



进行连接，在发送区发送16进制数据：01 01 00 00 00 06 01 03 00 00 00 04，则在接收区接收到从机数据：01 01 00 00 00 0b 01 03 08 ce 20 ce 30 ce 40 ce 50

其中 01 01 00 00 为复制主机的值

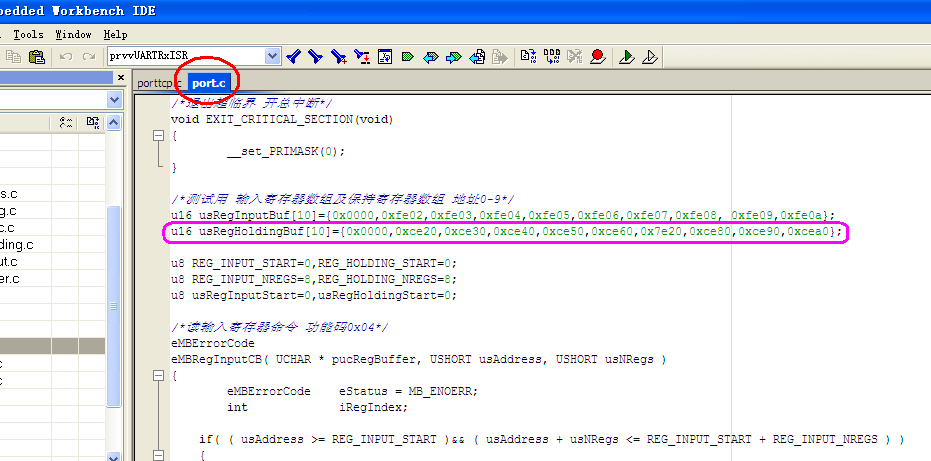
00 0b 为长度：11

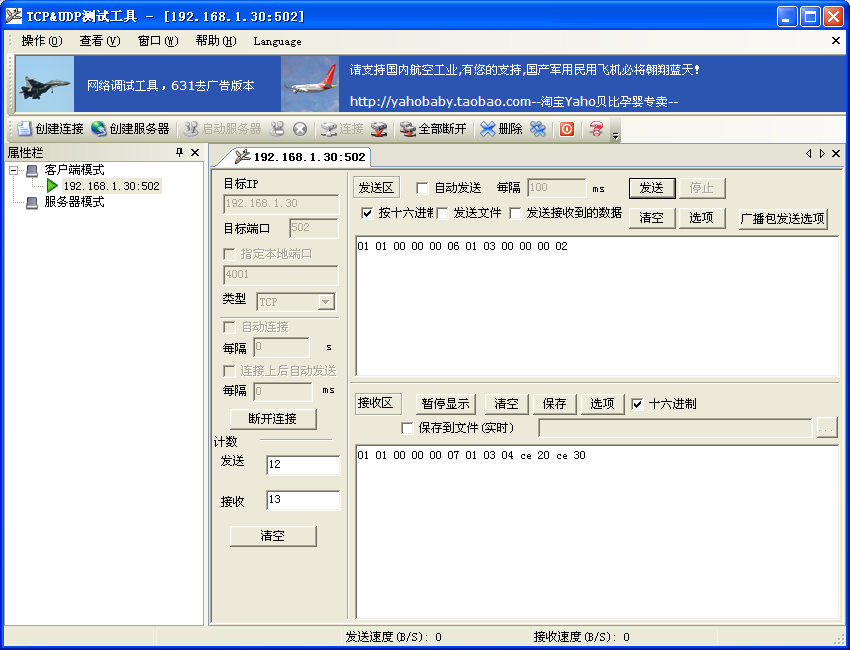
01 为表地址

03为功能码

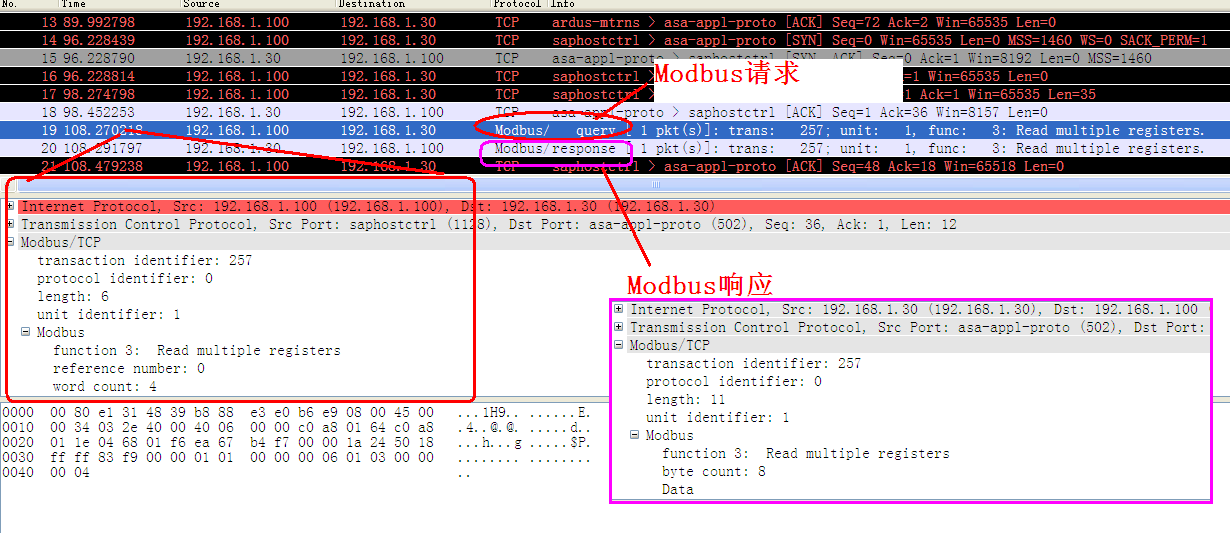
08 ce 20 ce 30 ce 40 ce 50为从机应答的保持寄存器值

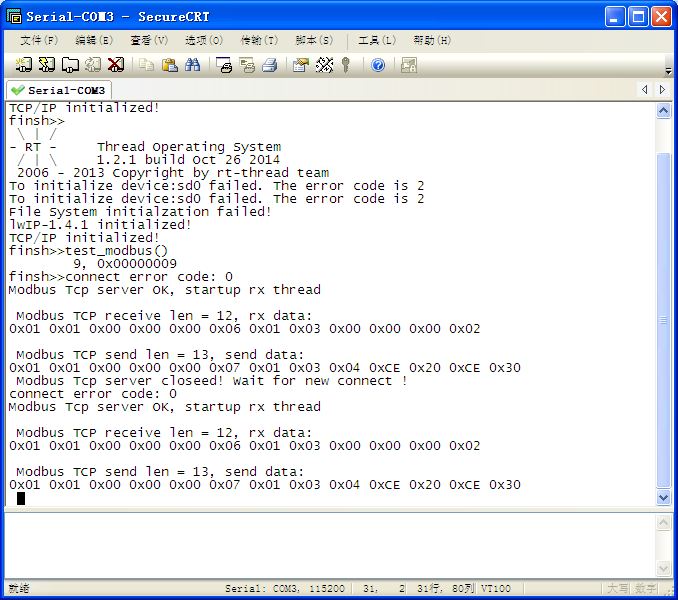
该值在port.c中定义：





WireShark 抓包结果：





如果连接断开，会提示：Modbus Tcp server closeed! Wait for new connect !

如果重新再次连接，会提示：

connect error code: 0

Modbus Tcp server OK, startup rx thread

检测连接断开，并重新连接在进程mbtcp\_srv\_thread中处理，当 ( !((netconn\_err(newconn) != ERR\_OK )||((checkethlink()!= ERR\_OK)))) 检测到网络断开，会断开当前连接，并重新准备接受新的连接请求：net\_acp\_result = netconn\_accept(conn,&newconn);