**功能要求：**

编写Modbus Master程序，实现对modbus slave开关量、浮点（温度数据）和整形设备参数、测量值的查询读取功能，以及对开关量、设备参数的设置功能。

Modbus协议概述

Modbus是由Modicon在1979年开发的，是全球第一个真正用于工业现场的总线协议。它是一个工业通信系统，也是是应用于电子控制器上的一种通用语言。此协议定义了一个控制器能认识使用的消息结构，而不管它们是经过何种网络进行通信的。它描述了一个控制器请求访问其它设备的过程，如何回应来自其它设备的请求，以及怎样侦测错误并记录，因此可应用于各种数据采集和过程监控。ModBus网络只有一个主机，所有通信都由他发出。网络可支持247个之多的远程从属控制器，但实际所支持的从机数要由所用通信设备决定。采用这个系统，各PC可以和中心主机交换信息而不影响各PC执行本身的控制任务。此协议支持传统的RS-232、RS-422、RS-485和以太网设备。

此项目采用modbus RTU格式代码，其采用8位二进制，（十六进制数0...9，A...F）一个字节表示一个数据，消息中的每个8位域都是一或两个十六进制字符组成每个字节组成为1个起始位，8个数据位，1或者0个奇偶校验位，无校验则无1个停止位（有校验时），2个Bit（无校验时）。当控制器设为在Modbus网络上以RTU模式通信，在消息中的每个8Bit字节按照原值传送，不做处理，如63H，RTU将直接发送01100011。这种方式的主要优点是：数据帧传送之间没有间隔，相同波特率下传输数据的密度要比ASCII高，传输速度更快 。

MODBUS数据单元

MODBUS协议定义了一个与基础层无关的简单协议数据单元,特定总线或网络上的MODBUS协议映射能够在应用数据单元上引入一些附加域，通用MODBUS帧如图2-2所示。启动MODBUS事务处理的客户机创建MODBUS PDU，其中的功能码向服务器指示将执行哪种操作，功能码后面是含有请求和响应参数的数据域。消息帧的地址域包含两个字符（ASCII）或8Bit（RTU）。可能的从设备地址是0…247 (十进制)。单个设备的地址范围是1…247。数据域是由两个十六进制数集合构成的，范围00…FF。根据网络传输模式，这可以是由一对ASCII字符组成或由一RTU字符组成。



**通用MODBUS帧**

MODBUS通信原理

MODBUS是一种简单的仿“客户机/服务器”型应用协议，其实质就是主/从模式，不过主机是客户机，从机是服务器。其通信遵循以下的过程：

（1）客户端准备请求并向服务器发送请求；

（2）服务器分析并处理客服端的请求，然后向客户端发送结果；

（3）如果出现任何差错，服务器将返回一个异常功能码。

在无差错的通信过程如下。

  **MODBUS通信**

功能实现分析

根据要求，要实现的主机的基础功能及其对应功能码有：

01 (0x01) 读线圈（开关量）

02 (0x02) 读只读线圈（开关量）

03 (0x03) 读保持寄存器

04 (0x04) 读输入（只读）寄存器

05 (0x05) 写单个线圈

06 (0x06) 写单个寄存器

15 (0x0F) 写多个线圈

16 (0x10) 写多个寄存器

RTU数据帧格式为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 从机 | 功能代码 | 设备地址 | 数据 | CRC校验 | 结束符 |
| 1字节 | 1字节 | 2字节 | n字节 | 2字节 | - |

要依据此格式，将软件用户设置信息、设定值转化为从机号、设备地址、功能代码、数据（字节类型），并根据传输数据产生CRC校验码。合并发送给从机，从而得到从机响应。

在接受从机报文时，也应当根据不同功能码下数据顺序特点，将报文“译码”，从而正确地将数据显示到用户界面。

编程实现

**主机发送报文**

**Ⅰ固定格式**

发送报文的前四个字节可以通过将设置界面内容转化（图1）而获取，在子窗体中将设置信息转化为int类型。而实现发送功能的函数将int类型数据转化为字节类数据，程序定义了ConvertIntToByteArray函数来实现此功能，转化结果合并到发送的字节数组报文中。值的注意的是，发送地址的数据需要减一，因为modbus协议默认从0开始记地址。

转化函数如下：

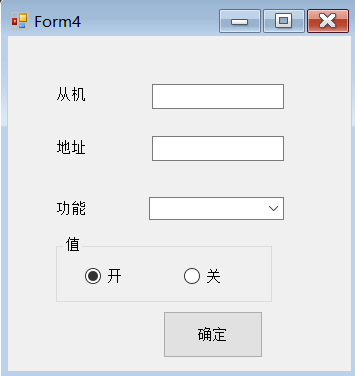
static void ConvertIntToByteArray(Int16 m, ref byte[] \_array)

{

\_array[0] = (byte)(m & 0xFF);

\_array[1] = (byte)((m & 0xFF00) >> 8);

}



**（1） （2）**

**Ⅱ数据部分**

01、02

对于主机来说，实现01和02功能码，报文中除了第二个表示功能码的字节内容不同，其他字节内容并无太大差异，实现方法类似。01、02功能地址区数据表示从那一个线圈开始读，而数据区是欲读线圈的个数，只需要将设置界面“长度”选项框内容转化为字节类型，合并发送。合并发送报文的字节数组为：

byte[] data = { \_slave[0], 0x01, \_address[1], \_address[0], \_length[1], \_length[0] }

03、04

同理，实现03和04功能码也很类似，区别在于数据区的内容转变为寄存器的数量。

合并转发报文的字节数组为：

byte[] data = { slaveb[0], 0x03, addressb[1], addressb[0], lengthb[1],lengthb[0] };

05、06

05功能码发送置单个线圈状态请求，地址区内容为欲置线圈的地址，数据区 内容为“FF00”或者“0000”，“FF00”表示请求置1，“0000”表示请求置0.

合并转发报文字节数组为：

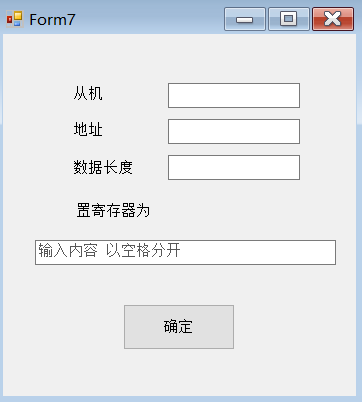
byte[] data = { slaveb[0], 0x05, addressb[1], addressb[0], \_onoff, 0x00 };

其中\_on0ff为字节类型，其值取决于置线圈设置窗口（2）中radiobutton选项。为FF或00，用if else语句实现.

06功能码为置单个寄存器请求，设置窗体如（3）地址为所置寄存器的地址，数据内容为欲置内容。一个寄存器内两个字节，因此处理它的方法和处理地址的方法相同，将其预置值转化为两个字节后组成数组。

合并转发报文字节数组为：

byte[] data = { slaveb[0], 0x06, addressb[1], addressb[0], valueb[1], valueb[0] };



**（3） （4）**

15、16

15功能码要求发出符合协议规范的置多个线圈的信息，地址字节和线圈数目字节内容和01功能码相同。但数据区的第一个字节要求给字节数，字节数是根据预置线圈的数目决定的。一个字节（八位）可以对应八个线圈的状态，如果整数个字节不能满足要求，则字节向上取整。数据区之后的内容则是预置线圈的信息，排序方式如下：

低地址线圈字节在先

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 08线圈 | 07线圈 | 06线圈 | 05线圈 | 04线圈 | 03线圈 | 02线圈 | 01线圈 |

高地址线圈字节在后

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 16线圈 | 15线圈 | 14线圈 | 13线圈 | 12线圈 | 11线圈 | 10线圈 | 09线圈 |

置位信息的转化实现使用了bitarray类型数据，先将输入的预置信息“01 00 01….”（字符串）按空格分割、计数并转化为int类型，再通过函数toboolean（大于0为1）转化为位信息，最后经如下的循环移位相加组成发送的字节数据。

data[i] = Convert.ToByte(data[i] \* 0x02 + Convert.ToInt16(mybit[i \* 8 + 7 - m]));

16功能码要求置多个寄存器，实现方法与15功能码类似，将输入内容分割、计数，最后转化为字节数组，但是16功能不必实行较为复杂的位操作，可如下分寄存器转化数据：

for (int i = 0; i < length16; i++)

{

byte[] shujv = new byte[2];

ConvertIntToByteArray(myint[i], ref shujv);

data[7 + i \* 2] = shujv[1]

data[8 + i \* 2 ] = shujv[0];

}

最后循环写入预置字节内容，使用重载serialwrite（byte[]发送内容，int起始字节，int长度）函数发送报文。

**Ⅲ校验部分**

RTU模式下采用CRC-16校验，将之前产生的数据的每一个字节按顺序与生成多项式循环右移异或，即做二进制除法，之后返回两个字节的数据。

**接收从机报文**

主机接受报文主要分为三个类型，通过设置界面改变标志参数，从而决定接受的模式

Ⅰ接受线圈信息，根据线圈回应报文的格式，第三个字节为发送给主机的字节数，按照此字节数循环接收数据并转化为bitarray类型，最后按位转为0/1字符在主界面显示。

Ⅱ接受整型信息，一个整型数据占两个字节，低位字节在前，高位字节在后。因此采用如下从第三个字节开始的循环接收，以两个字节为单位进行转化，得到每个寄存器的数据。

for (int i = 0; i < m; i++)

{

\_int[0] = receivedData[3 + i \* 2];

\_int[1] = receivedData[3 + i \* 2 + 1];

int zhi = Convert.ToInt16(receivedData[3 + i \* 2]) \* 256 + Convert.ToInt16(receivedData[3 + i \* 2 + 1]);//高8位乘256再相加

}

Ⅲ接收浮点数类型。单个浮点数由四个字节构成，从机发送时顺序为低位高八位、低位低八位、高位高八位、高位低八位，因此要按照以下方式接收组成字节数组才能正常转化为浮点数。（计算机处于小端模式），程序其他内容与接受整形数据基本相同。

\_float[1] = receivedData[3 + i \* 4];

\_float[0] = receivedData[3 + i \* 4 + 1];

\_float[3] = receivedData[3 + i \* 4 + 2];

\_float[2] = receivedData[3 + i \* 4 + 3];

**数据可视化及log保存**

数据可视化方面应用了用了chart控件，用于显示六个温度仪表的温度数据。在功能框体内选择仪表后，可显示某一个仪表6s内的温度数据。图形采用柱状图，程序后台根据从机报文刷新数据，用户可点击“实时”按钮就可将界面刷新为当前6s内的仪表数据。

程序还可以以.TXT格式保存主机界面中的内容。

**程序流程图**

