目录

第一章　MODBUS 协议简介

MODBUS 协议详细定义了校验码、数据序列等，这些都是特定数据交换的必要内容。

MODBUS 协议在一根通讯线上使用主从应答式连接（半双工），这意味着在一根单独的通讯线上信号沿着相反的两个方向传输。首先，主计算机的信号寻址到一台唯一的终端设备（从机），然后，终端设备发出的应答信号以相反的方向传输给主机。

MODBUS 协议只允许在主计算机和终端设备之间通讯，而不允许独立的设备之间的数据交换，这样各终端设备不会在它们初始化时占据通讯线路，而仅限于响应到达本机的查询信号。

1.1传输方式

传输方式是一个数据帧内一系列独立的数据结构以及用于传输数据的有限规则，下面定义了与 MODBUS 协议－ RTU 方式相兼容的传输方式。

- **Coding System**　二进制编码　**8** 位
- **Start bit**　起始位　**1** 位
- **Data bits**　数据位　**8** 位
- **Parity**　校验　无奇偶校验
- **Stop bit**　停止位　**1** 位
- **Error checking**　错误检测　**CRC**（循环冗余校验）

[注]瑞士 DAE 公司的网络电力仪表响应查询的时间为 0.1 ~ 1.0 秒（典型值为 0.4 秒）响应查询的时间指主机命令发出到收到回复的时间。

1.2 协议

当数据帧到达终端设备时，它通过一个简单的“端口”进入被寻址到的设备，该设备去掉数据帧的“信封”（数据头），读取数据，如果没有错误，就执行数据所请求的任务，然后，它将自己生成的数据加入到取得的“信封”中，把数据帧返回给发送者。返回的响应数据中包含了以下内容：终端从机地址(Address)、被执行了的命令(Function)、执行命令生成的被请求数据(Data)和一个校验码(Check)。发生任何错误都不会有成功的响应。

1.2.1 数据帧格式

Address	*Function*	*Data*	*Check*
8-Bits	8-Bits	N x 8-Bits	16-Bits

图 1 – 1 .数据帧格式

1.2.2 地址（Address）域

地址域在帧的开始部分，由一个字节 8 位（0 ~ 255）组成，这些位标明了用户指定的终端设备的地址，该设备将接收来自与之相连的主机数据。每个终端设备的地址必须是唯一的，仅仅被寻址到的终端会响应包含了该地址的查询。当终端发送回一个响应，响应中的从机地址数据便告诉了主机哪台终端正与之进行通信。

1.2.3 功能（Function）域

功能域代码告诉了被寻址到的终端执行何种功能。表 1 – 1 列出了瑞士 DAE 网络电力仪表用到的功能码，以及它们的意义和功能。

表 1 – 1 功能码

代码	意义	行为
01	读 DO 状态	获得数字（继电器）输出的当前状态（ON/OFF）
02	读 DI 状态	获得数字输入的当前状态（ON/OFF）
03	读数据寄存器	获得一个或多个寄存器的当前二进制值
05	控制 DO	控制数字（继电器）输出状态（ON/OFF）
16	预置多寄存器	设定二进制值到一系列多寄存器中

1.2.4 数据(Data)域

数据域包含了终端执行特定功能所需要的数据或者终端响应查询时采集到的数据。这些数据的内容可能是数值、参考地址或者设置值。例如：功能域码告诉终端读取一个寄存器，数据域则需要指明从哪个寄存器开始及读取多少个数据，内嵌的地址和数据依照类型和从机之间的不同内容而有所不同。

1.2.5 错误校验(Check)域

该域允许主机和终端检查传输过程中的错误。有时，由于电噪声和其它干扰，一组数据在从一个设备传输到另一个设备时在线路上可能会发生一些改变，出错校验能够保证主机或者终端不去响应那些传输过程中发生了改变的数据，这就提高了系统的安全性和效率，出错校验使用了 16 位循环冗余的方法（CRC16）。

[注] 发送序列总是相同的 – 地址、功能码、数据和与方向相关的出错校验。

1.3 错误检测

循环冗余校验（CRC）域占用两个字节，包含了一个 16 位的二进制值。CRC 值由传送设备计算出来，然后附加到数据帧上，接收设备在接收数据时重新计算 CRC 值，然后与接收到的 CRC 域中的值进行比较，如果这两个值不相等，就发生了错误。

CRC 运算时，首先将一个 16 位的寄存器预置为全 1，然后连续把数据帧中的每个字节中的 8 位与该寄存器的当前值进行运算，仅仅每个字节的 8 个数据位参与生成 CRC，起始位和终止位以及可能使用的奇偶位都不影响 CRC。

在生成 CRC 时，每个字节的 8 位与寄存器中的内容进行异或，然后将结果向低位移位，高位则用“0”补充，最低位（LSB）移出并检测，如果是 1，该寄存器就与一个预设的固定值（0A001H）进行一次异或运算，如果最低位为 0，不作任何处理。

上述处理重复进行，直到执行完了 8 次移位操作，当最后一位（第 8 位）移完以后，下一个 8 位字节与寄存器的当前值进行异或运算，同样进行上述的另一个 8 次移位异或操作，当数据帧中的所有字节都作了处理，生成的最终值就是 CRC 值。

生成一个 CRC 的流程为：

1、 预置一个 16 位寄存器为 0FFFFH（全 1），称之为 CRC 寄存器。

2、 把数据帧中的第一个字节的 8 位与 CRC 寄存器中的低字节进行异或运算，结果存回 CRC 寄存器。

3、 将 CRC 寄存器向右移一位。

4、 如果最低位为 0：重复第三步（下一次移位）。

如果最低位为 1：将 CRC 寄存器与一个预设的固定值（1010 0000 0000 0001）进行异或运算。

5、 重复第三步和第四步直到 8 次移位。这样处理完了一个完整的八位。

6、 重复第 2 步到第 5 步来处理下一个八位，直到所有的字节处理结束。

7、 最终 CRC 寄存器得值就是 CRC 的值。

8、 CRC 寄存器发送时将高位、低位互换。

注：

CRC 校验可用插表法和运算法两种方法实现，具体范例请看附录。

第二章　MODBUS 功能详解

本章的目标是为使用 DAE 网络电力仪表构造系统的程序员定义特定有效命令的通用格式，在每条数据查询格式说明的后面有一个该数据查询所执行的功能的解释和一个例子。

第一章已经简述了协议和数据帧，使用 DAE 网络电力仪表的程序员可以使用下述的方法以便通过协议正确地建立与它们通讯的特定应用程序。

本章所述协议将尽可能的使用如图 2 － 1 所示的格式，（数字为 16 进制）。

Addr	Fun	Data start reg hi	Data start reg lo	Data #of regs hi	Data #of regs lo	CRC16 Hi	CRC16 Lo
06H	03H	00H	00H	00H	21H	84H	65H

注：

Addr:从机地址

Fun:功能码

Data start reg hi：数据起始地址 寄存器高位

Data start reg lo：数据起始地址 寄存器低位

Data #of reg hi：数据读取个数 寄存器高位

Data #of reg lo：数据读取个数 寄存器低位

CRC16 Hi: 循环冗余校验 高位

CRC16 Lo: 循环冗余校验 低位

图 2 － 1 协议例述

2.1 读数字输出状态（功能码 01）

查询数据帧

此功能允许用户获得指定地址的从机控制的特定地址的 DO 输出状态 ON/OFF（1 = ON , 0 = OFF），除了从机地址和功能域，数据帧还需要在数据域中包含将被读取 DO 的初始地址和要读取的 DO 数量。SRTU510 中 DO 的地址从 0000H 开始（DO1=0000H，DO2=0001H）。

图 2 － 2 的例子是从地址为 17 的从机读取 DO1 到 DO6 的状态。

（例如：有 6 个 DO，DO 的地址应该为 0001~0006）

Addr	Fun	DO start reg hi	DO start reg lo	DO #of regs hi	DO #of regs lo	CRC16 Hi	CRC16 Lo
11H	01H	00H	00H	00H	06H	BEH	98H

图 2 － 2 读 DO1~DO6 的查询数据帧

响应数据帧

响应包含从机地址、功能码、数据的数量和 **CRC** 错误校验，数据包中每个 **DO** 占用一位（**1 = ON , 0 = OFF**），第一个字节的最低位为寻址到的 **DO** 值，其余的在后面。

图 **2 – 3** 所示为读数字输出状态响应的实例。

（**DO1 = OFF , DO2=ON,DO3=OFF,DO4=ON,DO5=OFF,DO6 = ON**）

Addr	Fun	Byte count	Data	CRC16 hi	CRC16 lo
11H	01H	01H	2AH	D4H	97H

DO 状态

0	0	DO6	DO5	DO4	DO3	DO2	DO1
0	0	1	0	1	0	1	0

MSB **LSB**

图 **2 – 3** 读 **DO1~DO6** 状态的响应数据帧

2.2 读数字输入状态（功能码 **02**）

查询数据帧

此功能允许用户获得 **DI** 的状态 **ON / OFF**（**1 = ON , 0 = OFF**），除了从机地址和功能域，数据帧还需要在数据域中包含将被读取 **DI** 的初始地址和要读取的 **DI** 数量。**SRTU510** 中 **DI** 的地址从 **0000H** 开始（**DI1=0000H**，**DI2=0001H** 依此类推）。具体地址请查看第三章。

图 **2 – 4** 的例子是从地址为 **17** 的从机读取 **DI1** 到 **DI16** 的状态。

（例如：**SRTU510** 有 **16** 个 **DI**，**DI** 的数量为 **1~16**）

Addr	Fun	DI start addr hi	DI start addr lo	DI num hi	DI num lo	CRC16 hi	CRC16 lo
11H	02H	00H	00H	00H	10H	7BH	56H

图 **2 – 4** 读 **DI1** 到 **DI16** 的查询

响应据数帧

响应包含从机地址、功能码、数据的数量和 **CRC** 错误校验，数据帧中每个 **DI** 占用一位（**1 = ON , 0 = off**），第一个字节的最低位为寻址到的 **DI** 值，其余的在后面。

图 **2-5** 所示为读数字输出状态**(DI1=on, DI2=on, DI3=off, DI4=off , DI5=on, DI6=on, DI7=off, DI8=off , DI9=off, DI10=off, DI11=on, DI12=on, DI13=off, DI14=off, DI15=on, DI16=on)**响应的实例。

Addr	Fun	Byte count	Data1	Data2	CRC16 hi	CRC16 lo
11H	02H	02H	33H	CCH	6CH	DEH

Data 1

DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1
0	0	1	1	0	0	1	1

MSB LSB

Data 2

DI16	DI15	DI14	DI13	DI12	DI11	DI10	DI9
1	1	0	0	1	1	0	0

MSB LSB

图 2 – 5 读 DI1 到 DI16 状态的响应

2.3 读数据（功能码 03）

查询数据帧

此功能允许用户获得设备采集与记录的数据及系统参数。

图 2 – 6 的例子是从 17 号从机（EPM420）读 3 个采集到的基本数据（数据帧中每个地址占用 2 个字节）U1,U2,U3, EPM420 中 U1 的地址为 0000H, U2 的地址为 0001H, U3 的地址为 0002H,

Addr	Fun	Data start addr hi	Data sta addr lo	Data #of regs h	Data #o regs	CRC16 hi	CRC16 lo
11H	03H	00H	00H	00H	03H	07H	5BH

图 2 – 6 读 U1、U2、U3 的查询数据帧

响应数据帧

响应包含从机地址、功能码、数据的数量和 CRC 错误校验。

图 2 – 6 的例子是读取 U1,U2,U3(U1=03E8H,U2=03E7H,U3=03E9H)的响应。

Addr	Fun	Byte count	Data1 hi	Data1 Lo	Data 2 hi	Data2 lo	Data3 hi	Data3 lo	CRC16 hi	CRC16 lo
11H	03H	06H	03H	E8H	03H	E7H	03H	E9H	FDH	9CH

图 2 – 7 读 U1,U2,U3 的响应数据帧

2.4 控制 DO （功能码 05）

查询数据帧

该数据帧强行设置一个独立的 DO 为 ON 或 OFF，DAE 公司产品的内部 DO 有的以集电极开路方式输出，有的以继电器输出，有的还可以选择电平方式（LATCH）或脉冲方式（PULSE）方式输出，具体使用请参考产品手册。SRTU 系列产品的 DO 的地址从 0000H 开始（DO1 = 0000H，DO2 = 0001H）。

数据 FF00H 将设 DO 为 ON 状态，而 0000H 则将设 DO 为 OFF 状态；所有其它的值都被忽略，并且不影响 DO。

下面的例子是请求 17 号从机设置 DO1 为 ON 状态。

Addr	Fun	DO addr hi	DO addr Lo	Value Hi	Value lo	CRC16 hi	CRC16 lo
11H	05H	00H	00H	FFH	00H	8EH	AAH

图示 2-8 控制独立的 DO 查询

响应数据帧

对这个命令请求的正常响应是在 DO 状态改变以后回传接收到的数据。

Addr	Fun	Do addr Hi	Do addr Lo	Value Hi	Value Lo	CRC16 Hi	CRC16 Lo
11H	05H	00H	00H	FFH	00H	8EH	AAH

图示 2-9 控制独立 DO 的响应响应

2.5 预置多寄存器（功能码 16）

查询数据帧

功能码 16(十进制)（十六进制为 10H）允许用户改变多个寄存器的内容，DAE 产品的系统内部的许多寄存器都可以使用此命令来改变其值。

注意：禁止对不具有可写属性的单元使用此命令改写。

下面的例子是预置 17 号从机(EPM420 数据帧中每个地址为 16 位存储) 正向有功电度 EP+为 178077833Kwh。 EP+的地址是 0040H， EP+占用 32 位，共 4 个字节。

Addr	Fun	Data sta reg hi	Data sta reg l	Data #o reg hi	Data #o reg lo	Byte Count	Value hi	Value Lo	Value hi	Value lo	CRC hi	CRC lo
11H	10H	00H	40H	00H	02H	04H	40H	89H	0AH	9DH	A0H	7CH

图示 **2-10** 预置 **EP+**

响应数据帧

对于预置单寄存器请求的正常响应是在寄存器值改变以后将接收到的数据传送回去。

Addr	Fun	Data star reg hi	Data star reg lo	Data #of reg hi	Data #of Reg lo	CRC16 hi	CRC16 lo
11H	10H	00H	40H	00H	02H	42H	8CH

图示 **2-11** **EP+**响应

附录 1：CRC 校验程序范例（DOS,C++）

```
#include "stdio.h"

static unsigned char revcrc16hi[]={
    0x00,0xC1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x00,0xc1,0x81,0x40,0x01,0xc0,0x80,0x41,
    0x01,0xc0,0x80,0x41,0x00,0xc1,0x81,0x40
};
static unsigned char revcrc16lo[]={
    0x00,0xc0,0xc1,0x01,0xc3,0x03,0x02,0xc2,
    0xc6,0x06,0x07,0xc7,0x05,0xc5,0xc4,0x04,
    0xcc,0x0c,0x0d,0xcd,0x0f,0xcf,0xce,0x0e,
    0x0a,0xca,0xcb,0x0b,0xc9,0x09,0x08,0xc8,
    0xd8,0x18,0x19,0xd9,0x1b,0xdb,0xda,0x1a,
    0x1e,0xde,0xdf,0x1f,0xdd,0x1d,0x1c,0xdc,
    0x14,0xd4,0xd5,0x15,0xd7,0x17,0x16,0xd6,
```

```
    0xd2,0x12,0x13,0xd3,0x11,0xd1,0xd0,0x10,
    0xf0,0x30,0x31,0xf1,0x33,0xf3,0xf2,0x32,
    0x36,0xf6,0xf7,0x37,0xf5,0x35,0x34,0xf4,
    0x3c,0xfc,0xfd,0x3d,0xff,0x3f,0x3e,0xfe,
    0xfa,0x3a,0x3b,0xfb,0x39,0xf9,0xf8,0x38,
    0x28,0xe8,0xe9,0x29,0xeb,0x2b,0x2a,0xea,
    0xee,0x2e,0x2f,0xef,0x2d,0xed,0xec,0x2c,
    0xe4,0x24,0x25,0xe5,0x27,0xe7,0xe6,0x26,
    0x22,0xe2,0xe3,0x23,0xe1,0x21,0x20,0xe0,
    0xa0,0x60,0x61,0xa1,0x63,0xa3,0xa2,0x62,
    0x66,0xa6,0xa7,0x67,0xa5,0x65,0x64,0xa4,
    0x6c,0xac,0xad,0x6d,0xaf,0x6f,0x6e,0xae,
    0xaa,0x6a,0x6b,0xab,0x69,0xa9,0xa8,0x68,
    0x78,0xb8,0xb9,0x79,0xbb,0x7b,0x7a,0xba,
    0xbe,0x7e,0x7f,0xbf,0x7d,0xbd,0xbc,0x7c,
    0xb4,0x74,0x75,0xb5,0x77,0xb7,0xb6,0x76,
    0x72,0xb2,0xb3,0x73,0xb1,0x71,0x70,0xb0,
    0x50,0x90,0x91,0x51,0x93,0x53,0x52,0x92,
    0x96,0x56,0x57,0x97,0x55,0x95,0x94,0x54,
    0x9c,0x5c,0x5d,0x9d,0x5f,0x9f,0x9e,0x5e,
    0x5a,0x9a,0x9b,0x5b,0x99,0x59,0x58,0x98,
    0x88,0x48,0x49,0x89,0x4b,0x8b,0x8a,0x4a,
    0x4e,0x8e,0x8f,0x4f,0x8d,0x4d,0x4c,0x8c,
    0x44,0x84,0x85,0x45,0x87,0x47,0x46,0x86,
    0x82,0x42,0x43,0x83,0x41,0x81,0x80,0x40
};
```

查表法子程序

```
unsigned short CRC16(unsigned char  *pointMsg,unsigned short DataLen)
{
unsigned char CRC_Hi=0xff;
unsigned char CRC_Lo=0xff;
unsigned Index;
while(DataLen--)
    {
    Index=CRC_Hi^*pointMsg++;
    CRC_Hi=CRC_Lo^revcrc16hi[Index];
    CRC_Lo=revcrc16lo[Index];
    }
return(CRC_Hi<<8|CRC_Lo);
}
```

运算法子程序

```
unsigned int CheckCRC(unsigned char *pData,int nLen)
{
unsigned int temp=0xffff,temp1,i,j;
for(i=0;i<nLen;i++)
    {
```

```
    temp^=*(pData+i);
    for(j=0;j<8;j++)
        {
        temp1=temp;
        temp>>=1;
        if(temp1&0x0001)
            temp^=0xa001;
        }
    }
return(temp);
}


main()
{
unsigned char pData[]={0x00,0x00};
unsigned short nLen=2;
int i,j;
struct LH{unsigned char LOBYTE;unsigned char HIBYTE;};
union{struct LH cc1;unsigned int crc;}CC1;
union{struct LH cc2;unsigned int CRC;}CC2;
for(i=0;i<256;i++)
{
pData[0]=i;
for(j=0;j<256;j++)
    {
    pData[1]=j;
    CC1.crc=CRC16(pData,nLen);
    CC2.CRC=CheckCRC(pData,nLen);
    if((CC1.cc1.LOBYTE!=CC2.cc2.HIBYTE)||(CC1.cc1.HIBYTE!=CC2.cc2.LOBYTE))
        {
        printf("i=%u   j=%u    Wrong!\n",i,j);
        printf("    %2x--%2x    \n",CC1.cc1.HIBYTE,CC1.cc1.LOBYTE);
        printf("    %2x--%2x    \n\n",CC2.cc2.HIBYTE,CC2.cc2.LOBYTE);
        }
    }
}
printf(" OK!\n");
}
```