pH(or ORP)探头

MODBUS RTU

操作员手册

目录

1 MODBUS RTU 基本信息 3
1.1 概述 3
1.2 MODBUS 命令结构 3
1.3 MODBUS RTU 在 pH（或 ORP）探头中的实施 5
1.4 pH（或 ORP）探头的 MODBUS RTU 功能码 5
1.5 pH（或 ORP）探头中的数据格式 7
2 MODBUS RTU 中的 pH（或 ORP）探头命令 9
2.1 概述 9
2.2 各命令详述 9

1 MODBUS RTU 基本信息

1.1 概述

此文档适合的 pH（或 ORP）探头的硬件版本号为 V1.2；软件版本号为 **V1.7** 及以上版本。
本文档详细介绍了 pH（或 ORP）探头的 MODBUS RTU 接口，目标对象是软件程序员。

1.2 MODBUS 命令结构

本文档中的数据格式说明;

----二进制显示，后缀用 B，例如:10001B
----十进制显示，无任何前后缀，例如:256
----十六进制显示，前缀用 0x，例如:0x2A
----ASCII 字符或 ASCII 字符串显示，例如:" YL4314010022"

1.2.1 命令结构

MODBUS 应用协议定义了简单协议数据单元(PDU)，与基础通信层无关:

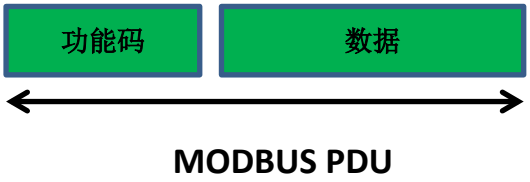


图 1: MODBUS 协议数据单元

特定总线或网络上的 MODBUS 协议映射介绍了协议数据单元的附加字段。启动 MODBUS 交换的客户端创建 MODBUS PDU;随后添加域，建立正确的通信 PDU.

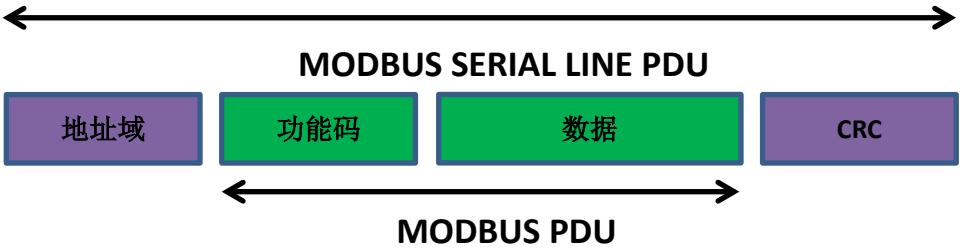


图 2: 串行通信的 MODBUS 结构

在 MODBUS 串行线上，地址域仅包含从设备地址。

提示:

pH（或 ORP）探头的从设备地址范围是 1…247

在主机发送的请求帧的地址域中设置从机的设备地址。从机回馈响应时，将自己的设备地址放置在响应帧的地址域中，使得主站知道哪个从机回馈响应的。

功能码指示服务器执行的操作类型。

CRC 域是"冗余校验"计算结果，按照信息内容执行。

1.2.2 MODBUS RTU 传输模式

设备使用 RTU(远程终端单元)模式进行 MODBUS 串行通信时，每条信息的 8 位字节包含两个 4 位十六进制字符。此模式的主要优点是具有更大的字符密度，比相同波特率的 ASCII 模式具有更好的数据吞吐量。每条信息必须以连续的字符串传输。

在 RTU 模式中的每个字节的格式(11 位):

编码系统: 8 位二进制

报文中每个 8 位字节含有两个 4 位十六进制字符(0-9、A-F)

每个字节中的位: 1 个起始位

8 个数据位，先发最低有效位

无奇偶校验位

2 位停止位

波特率: 9600bps

字符是如何串行传送的:

每个字符或字节均由此顺序发送(从左到右)

最低有效位(LSB)……最高有效位(MSB)

起始位	1	2	3	4	5	6	7	8	停止位

图 3: RTU 模式位序列

检查域结构:

循环冗余校验(CRC16)

结构说明:

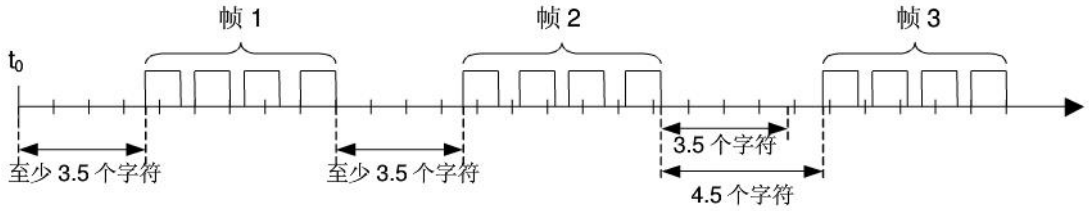
从设备地址	功能码	数据	CRC	
1 个字节	1 个字节	0…252 字节	2 个字节	
			CRC 低字节	CRC 高字节

图 4: RTU 信息结构

MODBUS 帧最大为 256 字节

1.2.3 MODBUS RTU 信息帧

在 RTU 模式，报文帧由时长至少为 3.5 个字符时间的空闲间隔区分，在后续部分，这个时间区间被称作 t3.5.



起始	地址	功能代码	数据	CRC 校验	结束
>=3.5 个字符	8 位	8 位	N×8 位	16 位	>=3.5 个字符

图 5: RTU 报文帧

整个报文帧必须以连续的字符流发送。

两个字符之间的停顿时间间隔超过 1.5 个字符时，信息帧认为不完整，接收方不接收此信息帧。

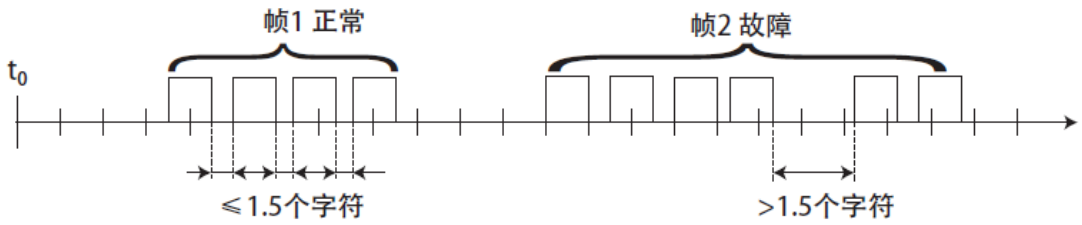


图 6: 帧的数据传输

1.2.4 MODBUS RTU CRC 校验

在 RTU 模式包含一个对全部报文内容执行的，基于循环冗余校验(CRC)算法的错误检测域。CRC 域检查整个报文的内容，不管报文有无奇偶校验，均执行此校验。CRC 域包含由两个 8 位字节组成的一个 16 位值。采用 CRC16 校验。低字节在前，高字节在后。

1.3 MODBUS RTU 在 pH（或 ORP）探头中的实施

根据官方 MODBUS 定义，由 3.5 个字符间隔触发命令开始，同样，命令结束也通过 3.5 个字符间隔表示。设备地址和 MODBUS 功能代码有 8 位。数据字符串包含 n×8 位，数据字符串包含寄存器的起始地址和读/写寄存器的数量。CRC 校验为 16 位。

	开始	设备地址	功能码	数据	总和校验		结束
数值	在 3.5 个字符期间无信号	1-247	符合 MODBUS 规范的功能码	符合 MODBUS 规范的数据	CRCL	CRCH	在 3.5 个字符期间无信号
字节	3.5	1	1	n	1	1	3.5

图 7: 数据传输的 MODBUS 定义

1.4 pH（或 ORP）探头的 MODBUS RTU 功能码

pH（或 ORP）探头仅使用两个 MODBUS 功能码:

0x03: 读保持寄存器 0x10： 写多重寄存器

1.4.1 MODBUS 功能码 0x03：读保持寄存器

此功能码用于读取远程设备的保持寄存器的连续块内容。请求 PDU 指定开始寄存器地址和寄存器数量。从零开始寻址寄存器。因此，寻址寄存器 1-16 为 0-15。响应信息中的寄存器数据按照每个寄存器两个字节打包。对于每个寄存器，第一个字节包含高位比特，第二个字节包含低位比特。

请求

功能码	1 个字节	0x03
开始地址	2 个字节	0x0000….0xffff
读取寄存器数量	2 个字节	1…125

图 8: 读取保持寄存器请求帧

响应

功能码	1 个字节	0x03
字节数	1 个字节	N×2
寄存器值	N×2 个字节	

N = 寄存器数量

图 9: 读取保持寄存器响应帧

下面以读取保持寄存器 108-110 为例说明请求帧和响应帧。(寄存器 108 的内容只读，为两个字节数值 0X022B，寄存器 109-110 内容为 0X0000 和 0X0064)

请求帧		响应帧	
数制	(十六进制)	数制	(十六进制)
功能码	0x03	功能码	0x03
开始地址(高字节)	0x00	字节数	0x06
开始地址(低字节)	0x6B	寄存器值(高字节) (108)	0x02
读取寄存器数量(高字节)	0x00	寄存器值(低字节) (108)	0x2B
读取寄存器数量(低字节)	0x03	寄存器值(高字节) (109)	0x00
		寄存器值(低字节) (109)	0x00
		寄存器值(高字节) (110)	0x00
		寄存器值(低字节) (110)	0x64

图 10: 读取保持寄存器请求帧和响应帧实例

1.4.2 MODBUS 功能码 0x10：写多重寄存器

此功能码用于向远程设备中写入连续寄存器(1…123 个寄存器)块，在请求数据帧中指定写入的寄存器值。数据以每个寄存器两个字节打包。响应帧返回功能码，开始地址和写入的寄存器的数量。

请求

功能码	1 个字节	0x10
开始地址	2 个字节	0x0000….0xffff
输入寄存器数量	2 个字节	0x0001….0x0078
字节数	1 个字节	N×2
寄存器值	N×2 个字节	值

N = 寄存器数量

图 11: 写多重寄存器请求帧

响应

功能码	1 个字节	0x10
开始地址	2 个字节	0x0000….0xffff
寄存器数量	2 个字节	1…123(0x7B)

N = 寄存器数量

图 12: 写多重寄存器响应帧

下面以写入数值 0x000A 和 0x0102 至开始地址为 2 的两个寄存器中为例说明请求帧和响应帧。

请求帧		响应帧	
数制	(十六进制)	数制	(十六进制)
功能码	0x10	功能码	0x10
开始地址(高字节)	0x00	开始地址(高字节)	0x00
开始地址(低字节)	0x01	开始地址(低字节)	0x01
输入寄存器数量(高字节)	0x00	输入寄存器数量(高字节)	0x00
输入寄存器数量(低字节)	0x02	输入寄存器数量(低字节)	0x02
字节数	0x04		
寄存器值(高字节)	0x00		

寄存器值(低字节)	0x0A	
寄存器值(高字节)	0x01	
寄存器值(低字节)	0x02	

图 13: 写多重寄存器请求帧和响应帧实例

1.5 pH（或 ORP）探头中的数据格式

1.5.1 浮点数

定义: 浮点数，符合 IEEE 754(单精度)

说明	符号	指数	尾数	总和
位	31	30…23	22…0	32
指数偏差	127			

图 14: 浮点数单精度定义(4 个字节，2 个 MODBUS 寄存器)

实例: 将十进制数 17.625 编译成二进制数

步骤 1：将十进制形式表示的 17.625 转换成二进制形式的浮点数

先求整数部分的二进制表示

$$17_{十进制} = 16 + 1 = 1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$$

所以整数部分 17 的二进制表示为 10001B

再求小数部分的二进制表示

$$0.625 = 0.5 + 0.125 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$

所以小数部分 0.625 的二进制表示为 0.101B

所以十进制形式表示的 17.625 的二进制形式的浮点数为 10001.101B

步骤 2：移位求指数。

将 10001.101B 向左移，直到小数点前只剩下一位，得到 1.0001101B，而 10001.101B = 1.0001101 B× 2^4。所以指数部分为 4，加上 127，变为 131，其二进制表示为 10000011B，

步骤 3：计算尾数

去除 1.0001101B 的小数点前的 1 得到尾数为 0001101B（因为小数点前必定为 1，所以 IEEE 规定只记录小数点后面的就可以),针对 23 位尾数的重要说明:第一位(即隐藏位)不编译。隐藏位是分隔符左侧的位，此位通常被设置为 1 并抑制。

步骤 4：符号位定义

正数的符号位为 0，负数的符号位为 1,所以 17.625 的符号位为 0.

步骤 5：转化为浮点数

1 位符号 + 8 位指数 + 23 位尾数

0 10000011 00011010000000000000000B(对应十六进制表示为 0x418D0000)

参考代码:

1、如果用户使用的编译器有实现此功能的库函数则可以直接调用此库函数，例如使用的是 C 语言，那么可以直接调用 C 库函数 memcpy 获取一个浮点数在内存中存储格式的整数表示;

例如:
```
float   floatdata;//被转化的浮点数
void*  outdata;
memcpy(outdata,&floatdata,4);
```

假如 floatdata=17.625
若为**小端存储模式**则执行完上面的语句后则
地址单元 outdata 存储的数据为 0x00
地址单元(outdata+1) 存储的数据为 0x00
地址单元(outdata+2) 存储的数据为 0x8D
地址单元(outdata+3) 存储的数据为 0x41
若为大端存储模式则执行完上面的语句后
地址单元 outdata 存储的数据为 0x41
地址单元(outdata+1) 存储的数据为 0x8D
地址单元(outdata+2) 存储的数据为 0x00
地址单元(outdata+3) 存储的数据为 0x00
2、如果用户使用的编译器没有实现此功能的库函数则可以用如下的函数实现此功能:

```
void memcpy(void *dest,void *src,int n)
{
    char *pd = (char *)dest;
    char *ps = (char *)src;
    for(int i=0;i<n;i++)  *pd++ = *ps++;
}
```

然后同上进行调用 memcpy(outdata,&floatdata,4);

实例: 将二进制浮点数 0100 0010 0111 1011 0110 0110 0110 0110B 编译为十进制数
步骤 1: 将二进制浮点数 0100 0010 0111 1011 0110 0110 0110 0110B 分为符号位、指数位和尾数位

0 10000100 11110110110011001100110B

1 位符号 + 8 位指数 + 23 位尾数
符号位 S: 0 表示正数
指数位 E：10000100B $=1\times2^7+0\times2^6+0\times2^5+0\times2^4+0\times2^3+1\times2^2+0\times2^1+0\times2^0$
$=128+0+0+0+0+4+0+0=132$
尾数位 M：11110110110011001100110B =8087142
步骤 2: 计算十进制数

$D=(-1)^S\times(1.0+M/2^{23})\times2^{E-127}$
$=(-1)^0\times(1.0+8087142/2^{23})\times2^{132-127}$
$=1\times1.964062452316284\times32$
$=62.85$

参考代码:

```
float floatTOdecimal(long int byte0, long int byte1, long int byte2, long int byte3)
{
    long int realbyte0,realbyte1,realbyte2,realbyte3;
    char S;
    long int  E,M;
    float D;
    realbyte0 = byte3;
    realbyte1 = byte2;
    realbyte2 = byte1;
```

```
        realbyte3 = byte0;

        if((realbyte0&0x80)==0)
        {
           S = 0;//正数
         }
         else
        {
           S = 1;//负数
        }
        E = ((realbyte0<<1)|(realbyte1&0x80)>>7)-127;
        M = ((realbyte1&0x7f) << 16) | (realbyte2<< 8)| realbyte3;
        D = pow(-1,S)*(1.0 + M/pow(2,23))* pow(2,E);
        return D;
}
```

函数说明:参数----byte0、byte1、byte2、byte3 代表二进制浮点数的 4 个字节(

返回值----转换得到的十进制数

例如用户向探头发送获取 pH 值命令，收到的应答帧中的代表 pH 值的 4 个字节为 0x33,0x33,0xF3,0x40,那么用户可以通过下面的调用语句得到对应的 pH 值的十进制数即 pH = 7.6。

```
float pH = floatTOdecimal( 0x33, 0x33, 0xF3, 0x40);
```

1.5.2 字符

定义: 字符的用 ASCII 码表示.

实例:字符串“YL”在命令传输中表示为对应的 ASCII 码(以下编译参考 ASCII 码表格)

“Y” 在命令传输中表示为 0x59

“L” 在命令传输中表示为 0x4C

2 MODBUS RTU 中的 pH（或 ORP）探头命令

2.1 概述

为了与 pH（或 ORP）探头电极进行 MODBUS RTU 通信，需要 MODBUS 主站终端应用软件。MODBUS RTU 是开放式标准，提供多个免费商业应用工具套件。在本手册中，MODBUS 寄存器的地址从 1 开始。但是，MODBUS 主站协议从寄存器地址 0 开始工作。通常，MODBUS 主站软件编译地址。因此，寄存器地址 2090 将被 MODBUS 主站软件编译为 2089。

2.2 各命令详述

2.2.1 设置从机 ID

作用：设置电极的 MODBUS 从设备地址，地址范围为 1~247。

可以通过地址为 0x3000 的 MODBUS 寄存器设置电极的 MODBUS 从设备地址。

起始地址	寄存器数量	寄存器 1	MODBUS 功能码
0x3000	0x01	新的设备地址	0x10

图 15: 设置从机 ID 命令寄存器定义

下面以电极旧的设备地址=0x01，新的设备地址=0x14 为例说明设置从机 ID 命令的请求帧和应答帧

定义	地址域	功能码	起始地址		寄存器数量		字节数	寄存器值		CRC	
字节	0	1	2	3	4	5	6	7	8	9	10
内容	0x01	0x10	0x30	0x00	0x00	0x01	0x02	0x14	0x00	0x99	0x53

图 16: 设置从机 ID 请求帧实例　　备注: byte8 为保留值，无意义

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x10	0x30	0x00	0x00	0x01	0x0E	0xC9

图 17: 设置从机 ID 应答帧实例

2.2.2 获取 SN

作用：获取电极的识别号 SN，每个电极都有惟一的 SN。

可以从起始地址为 0x0900 的连续 7 个 MODBUS 寄存器中读取探头的 SN。

起始地址	寄存器数量	寄存器 1—7	MODBUS 功能码
0x0900	0x07	SN	0x03

图 18: 获取 SN 命令寄存器定义

下面以从设备地址 0x01，返回的 SN“YL4314010022”为例说明获取 SN 命令的请求帧和应答帧

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x09	0x00	0x00	0x07	0x07	0x94

图 19: 获取 SN 命令请求帧实例

定义	地址域	功能码	字节数	寄存器值			CRC	
字节	0	1	2	3	4~15	16	17	18
内容	0x01	0x03	0x0E	0x00	“YL4314010022”	0x00	0xAD	0x9C

图 20: 获取 SN 命令应答帧实例　　备注:探头 SN 如下，以 ASCII 形式存储

字节	4	5	6	7	8	9	10	11	12	13	14	15
内容	0x59	0x4C	0x34	0x33	0x31	0x34	0x30	0x31	0x30	0x30	0x32	0x32

图 21: 探头的 SN

2.2.3 获取 pH 和电位值（ORP）

作用：获取探头的 pH 和电位值（ORP）：电位值（ORP）的单位为 mV。

可以从起始地址为 0x2600 的连续 4 个 MODBUS 寄存器中读取探头的温度和溶氧值。

起始地址	寄存器数量	寄存器 1、2	寄存器 3、4	MODBUS 功能码
0x2600	0x04	电位值（ORP）	pH 值	0x03

图 26: 获取温度和溶氧值命令的寄存器定义

下面以从设备地址 0x01,返回的 pH 值为 7，电位值（ORP）为-6.56mV 为例说明获取 pH 和电位值（ORP）命令的请求帧和应答帧。

定义	地址域	功能码	起始地址	寄存器数量	CRC

字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x26	0x00	0x00	0x04	0x4F	0x41

图 27: 获取 pH 值和电位值（ORP）命令的请求帧

定义	地址域	功能码	字节数	寄存器值		CRC	
字节	0	1	2	3~6	7~10	11	12
内容	0x01	0x03	0x08	-6.56	7	0x5C	0xE6

图 28: 获取 pH 值和电位值（ORP）命令的应答帧 备注: pH 值、电位值（ORP）：小端存储模式，浮点数

电位值（ORP）（3-6）				pH 值（7~10）			
0x85	0xEB	0xD1	0xC0	0x00	0x00	0xE0	0x40

图 29: pH 值和电位值（ORP）字节分布

2.2.4 获取电位值（ORP）

作用：校准 pH 时，获取其电位值（ORP），单位为 mV。

使用 MODBUS 寄存器 0x1200。

起始地址	寄存器数量	寄存器 1、2	MODBUS 功能码
0x1200	0x02	电位值（ORP）	0x03

图 30: 获取电位值（ORP）命令的寄存器定义

下面以从设备地址 0x01，返回的电位值（ORP）为-10.28 为例说明获取电位值（ORP）命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x12	0x00	0x00	0x02	0xC1	0x73

图 31: 获取电位值（ORP）命令的请求帧

定义	地址域	功能码	字节数	寄存器值	CRC	
字节	0	1	2	3~6	7	8
内容	0x01	0x03	0x04	-10.28	0x37	0x46

图 32: 获取电位值（ORP）命令的应答帧

电位值（3~6）			
0xE1	0x7A	0x24	0xC1

图 33: 电位值（ORP）字节分布

2.2.5 获取软件和硬件版本号

作用：获取当前使用的硬件版本号和软件版本号。

可以从起始地址为 0x0700 的连续 2 个 MODBUS 寄存器中读取探头的软件和硬件版本号。

起始地址	寄存器数量	寄存器 1	寄存器 2	MODBUS 功能码
0x0700	0x02	硬件版本号	软件版本号	0x03

图 34: 获取软件和硬件版本号命令的寄存器定义

下面以从设备地址 0x01，返回的硬件版本 1.1，软件版本 1.3 为例说明获取软件和硬件版本号命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x07	0x00	0x00	0x02	0xC5	0x7F

图 35: 获取软件和硬件版本号命令的请求帧

定义	地址域	功能码	字节数	寄存器值	CRC

字节	0	1	2	3~4		5~6		7	8
内容	0x01	0x03	0x04	0x01	0x01	0x03	0x01	0x6A	0x5F

图 36: 获取软件和硬件版本号命令的应答帧

2.2.6 获取温度值

作用：获取待测 pH 溶液当前温度值。

可以通过从起始地址为 0x2400 的连续 2 个 MODBUS 寄存器获取温度值。

起始地址	寄存器数量	寄存器 1、2	MODBUS 功能码
0x2400	0x02	温度	0x03

图 44: 获取温度值命令的寄存器定义

下面以从设备地址 0x01，返回的温度值为 15.8℃例说明获取温度值命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x24	0x00	0x00	0x02	0xCE	0xFB

图 45: 获取温度值命令的请求帧 备注: 温度值：小端存储模式，浮点数

定义	地址域	功能码	字节数	寄存器值	CRC	
字节	0	1	2	3~6	7	8
内容	0x01	0x03	0x04	15.8	0xE4	0x50

图 46: 获取温度值命令的应答帧

温度值（3~6）			
0xCD	0xCC	0x7C	0x41

图 47: 温度值字节分布

2.2.7 设置 pH 用户校准参数

作用：设置两个 pH 校准参数 K、B。

可以通过起始地址为 0x1100 的连续 4 个 MODBUS 寄存器设置 pH 校准参数。

起始地址	寄存器数量	寄存器 1、2	寄存器 3、4	MODBUS 功能码
0x1100	0x04	K 值	B 值	0x10

图 64: 设置 pH 校准参数命令的寄存器定义

下面以从设备地址 0x01 为例说明设置 pH 校准参数命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		字节数	寄存器值		CRC	
字节	0	1	2	3	4	5	6	7~10	11~14	15	16
内容	0x01	0x10	0x11	0x00	0x00	0x04	0x08	1.0	0.0	0x81	0xAE

图 65: 设置 pH 校准参数命令请求帧 备注: K、B：小端存储模式，浮点数

K(7~10)				B(11~14)			
0x00	0x00	0x80	0x3F	0x00	0x00	0x00	0x00

图 66: K、B 值的字节分布

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x10	0x11	0x00	0x00	0x04	0xC4	0xF6

图 67: 设置 pH 校准参数命令应答帧

2.2.8 获取 pH 用户校准参数

作用：获取 pH 校准参数，即 K、B 值。

可以从起始地址为 0x1100 的连续 4 个 MODBUS 寄存器中读取 pH 校准参数。

起始地址	寄存器数量	寄存器 1、2	寄存器 3、4	MODBUS 功能码
0x1100	0x04	K 值	B 值	0x03

图 68: 获取 pH 校准参数命令的寄存器定义

下面以从设备地址 0x01,返回的 K 值为 1，B 值为 0 为例说明获取 pH 校准参数命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	0x01	0x03	0x11	0x00	0x00	0x04	0x41	0x35

图 69: 获取 pH 校准参数命令的请求帧

定义	地址域	功能码	字节数	寄存器值		CRC	
字节	0	1	2	3~6	7~10	11	12
内容	0x01	0x03	0x08	1	0	0x9E	0x12

图 70: 获取 pH 校准参数命令的应答帧

K 值（3~6）				B 值（7-10）			
0x00	0x00	0x80	0x3F	0x00	0x00	0x00	0x00

图 71: pH 校准参数字节分布

2.2.9 获取从机 ID

作用：获取当前电极的 MODBUS 从设备地址， 该命令以 0xFF 作为固定地址域。

可以从起始地址为 0x3000 的 MODBUS 寄存器中读取当前电极的 MODBUS 从设备地址。

起始地址	寄存器数量	寄存器 1	MODBUS 功能码
0x3000	0x01	当前设备地址	0x03

图 72: 获取从机 ID 的寄存器定义

下面以返回的地址 0x03 为例说明获取从机 ID 命令的请求帧和应答帧。

定义	地址域	功能码	起始地址		寄存器数量		CRC	
字节	0	1	2	3	4	5	6	7
内容	**0xFF**	0x03	0x30	0x00	0x00	0x01	0x9E	0xD4

图 73: 获取从机 ID 命令的请求帧

定义	地址域	功能码	字节数	寄存器值		CRC	
字节	0	1	2	3	4	5	6
内容	0xFF	0x03	0x02	0x03	0x00(保留)	0x91	0x60

图 74: 获取从机 ID 命令的应答帧