Modbus-RTU_RS485 通讯协议

一、 概述

二、 硬件连接

三、 通讯配置

四、 通讯命令

1、读当前压力值与温度值命令(03H)

2、读参数命令(04H)

3、微调、复位与恢复出厂数据命令(06H)

4、写参数命令(10H)

五、 附录

1、波特率(BT)参数对照表

2、校验方式(0ddP)参数对照表

3、float 压力数据格式参数(FFT)对照表

4、通讯命令响应时间表

5、CRC16 计算方法

Modbus-RTU_RS485 通讯协议

一、概述

Modbus-RTU 智能压力变送器作为从机在 RS485 通讯网络中遵循 Modbus RTU 协议与主机通讯实现远程读压力值、温度值，读写参数、恢复数据等操作。通讯命令采用 Modbus RTU 协议子集，遵循 Modbus 通讯时序，CRC-16 校验，4.8K-115.2K 的波特率可选，通讯高效、可靠。

二、硬件连接

用标准串口数据线 + **RS232-RS485** 转换模块或 **USB-485** 模块连接 PC 与产品(表 1)：

产品	RS485 模块
A(RS485)	A
B(RS485)	B

表 1

三、通讯配置

- 数据传输方式：

  异步 10、11、12 位——1 位起始位，8 位数据位，1 位或 2 位停止位，无校验或 1 位校验位。默认无校验，支持奇校验或偶校验。

- 数据传输波特率：4.8K、9.6K、19.2K、38.4K、57.6K，115.2K。

  默认波特率为 19.2K，可设置。

- 地址：1～247(默认为 2，可设置)。

- 产品默认通讯参数

从机地址：2

波特率：19200

数据位：8

校验位：无校验

停止位：1

- 产品输出压力数据格式

Float 压力数据，float 数据默认 ABCD 大端模式。

- 消息帧格式(表 3)，详细信息参考国标 GBT 19582-2008。

起始	设备地址	功能码	数据	CRC16 校验	结束
T1-T2-T3-T4	8bit	8bit	n＊ 8bit	16bit	T1-T2-T3-T4

表 3

- 单字符传送位序列(表 4)：

起始位	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7	校验位	停止位

表 4

四、通讯命令

命令一览表(表 5)：

序号	功能码	起始地址	寄存器数量	字节数	寄存器值	命令描述
1	03H	0000H	0004H	无	无	读压力值与温度值，(表 6~表 8)
		0000H	0002H			读压力值
		0004H				读温度值
2	04H	0004H	0002H	无	无	读参数值-量程下限 (SLL),只读(表 9~表 11)
		0008H				读参数值-量程上限 (SLH)，只读
		0028H				读参数值- 波特率(BT)
		002CH				读参数值- 丛机地址(DE)
		0030H				读参数值- 校验方式(OddP)
		0034H				读参数值- 停止位(Stop)
		003CH				读参数值- 压力数据 Float 格式(FFT)
3	06H	0000H	无	无	0000H	PV 清 0(表 12~表 14)
					0001H	零点微调
					0002H	满点微调
		0002H	无	无	00DEH	软件复位
		0004H			00A5H	恢复出厂数据
4	10H	0028H	0002H	04H	float data	写参数值- 波特率(BT)(表 15~表 17)
		002CH				写参数值- 丛机地址(DE)
		0030H				写参数值- 校验方式(OddP)
		0034H				写参数值- 停止位(Stop)
		003CH				写参数值- 压力数据 Float 格式(FFT))

表 5

1、读当前压力值与温度值命令(03H 命令)

主机发送命令帧(表 6)：

从机地址	功能码	起始地址	寄存器数量	CRC 校验码	
XXH	03H	XXXXH	YYYYH	CRCL	CRCH

表 6

从机地址:1~247，下同。

起始地址：0000H 或 0004H，需要同时读出压力温度数据或仅读出压力数据时选择 0000H，仅需要读出温度数据时选择 0004H。

寄存器数量：0004H 或 0002H，需要同时读出压力温度数据时选择 0004H，需要分别读出压力与温度数据时选择 0002H。

CRC：校验码低字节在前，高字节在后，下同。CRC16 计算方法参考附录 4。

从机正确响应时回送命令帧(表 7)：

从机地址	功能码	字节数	float 数据	CRC 校验码	
XXH	03H	XXH	压力+温度、压力或温度数据	CRCL	CRCH

表 7

正确响应时从机根据主机不同的命令返回 压力+温度、压力或温度数据。

字节数：04H 或 08H，主机同时读出压力温度数据时丛机返回 08H，主机仅读出压力或温度数据时丛机返回 04H。

从机接收数据错误时回送命令帧(表 8)：

从机地址	功能码	错误码	CRC 校验码	
XXH	83H	XXH	CRCL	CRCH

表 8

错误码范围：00H~03H，0AH~22H。

00H： 校验码错误，下同。

01H： 功能码错误，下同。

02H： 起始地址错误，下同。

03H：寄存器数量错误，下同。

0AH~22H： 产品运行状态错误，详情参照“RS485 产品报警代码表”文档。

eg1.

从机地址为 2，压力值 100.500，温度值 25.5 时，主机同时读压力值与温度值命令帧：

02H 03H 00H 00H 00H 04H 44H 3AH

从机正确响应时回送命令帧:

02H 03H 08H 42H C9H 00H 00H 41H CCH 00H 00H 92H 75H

42C90000H 为压力值 100.500 的浮点数格式，41CC0000H 为温度值 25.5 的浮点数格式。

eg2.

从机地址为 2，压力值 100.500，温度值 25.5 时，主机仅读压力值命令帧：

02H 03H 00H 00H 00H 02H C4H 38H

从机正确响应时回送命令帧:

02H 03H 04H 42H C9H 00H 00H 0DH 75H

42C90000H 为压力值 100.500 的浮点数格式。

eg3.

从机地址为 2，压力值 100.500，温度值 25.5 时，主机仅读温度值命令帧：

02H 03H 00H 04H 00H 02H 85H F9H

从机正确响应时回送命令帧:

02H 03H 04H 41H CCH 00H 00H 1DH 30H

41CC0000H 为温度值 25.5 的浮点数格式。

2、读参数值命令(04H 命令)

主机发送命令帧(表 9)：

从机地址	功能码	起始地址	寄存器数量	CRC 校验码	
XXH	04H	XXXXH	0002H	CRCL	CRCH

表 9

起始地址：0004H、0008H、0028H、002CH、0030H、0034H、003CH，详情参照表 5。

从机正确响应时回送命令帧(表 10)：

从机地址	功能码	字节数	float 数据				CRC 校验码	
XXH	04H	04H	Byte1	Byte2	Byte3	Byte4	CRCL	CRCH

表 10

注意：“波特率”、“校验方式”、“Float 压力数据格式”参数映射关系参考附录 1~3。

从机接收数据错误时回送命令帧(表 11)：

从机地址	功能码	错误码	CRC 校验码	
XXH	84H	XXH	CRCL	CRCH

表 11

3、微调、软件复位与恢复出厂数据命令(06H 命令)

主机发送命令帧(表 12)：

从机地址	功能码	起始地址	寄存器值	CRC 校验码	
XXH	06H	XXXXH	YYYYH	CRCL	CRCH

起始地址：0000H、0002H、0004H，详情参照表 5。 表 12

寄存器值：参照表 5。

从机正确响应时回送命令帧(表 13)：

从机地址	功能码	起始地址	寄存器值	CRC 校验码	
XXH	06H	XXXXH	YYYYH	CRCL	CRCH

注意：对于复位命令，从机正确响应时直接复位，不向主机返回任何数据。 表 13

从机接收数据错误时回送命令帧(表 14)：

从机地址	功能码	错误码	CRC 校验码	
XXH	86H	XXH	CRCL	CRCH

表 14

4、写参数值命令(10H 命令)

主机发送命令帧(表 15)：

从机地址	功能码	起始地址	寄存器数量	字节数	float 型 参数数据				CRC 校验码	
XXH	10H	XXXXH	02H	04H	Byte1	Byte2	Byte3	Byte4	CRCL	CRCH

表 15

起始地址：0028H、002CH、0030H、0034H、003CH，详情参照表 5。

从机正确响应时回送命令帧(表 16)：

从机地址	功能码	起始地址	寄存器数量	CRC 校验码	
XXH	10H	XXXXH	0002H	CRCL	CRCH

表 16

从机接收数据错误时回送命令帧(表 17)：

从机地址	功能码	错误码	CRC 校验码	
XXH	90H	XXH	CRCL	CRCH

表 17

错误码 04H：字节数错误。

注意：

◆ “波特率”、“校验方式”、“Float 压力数据格式”参数值映射关系参考附录 1~3。

◆ “波特率”、“校验方式”、”停止位”参数被修改后，必须复位产品才可生效。

五、附录

附录 1——波特率(表 18)：

参数值	0	1	2	3	4	5
波特率	4800	9600	19200	38400	57600	115200

表 18

附录 2——校验方式(表 19)：

参数值	0	1	2
校验方式	无校验	奇校验	偶校验

表 19

附录 3——float 压力数据格式(表 20)：

参数值	0	1	2	3
Float 压力数据格式	ABCD	CDAB	BADC	DCBA

表 20

附录 4——通讯命令响应时间表(表 21)：

序号	通讯命令	响应时间	测试条件
1	读压力与温度值命令	≤50ms	波特率：19200 数据位：8 位 校验方式：无校验 停止位：1 位 注意：响应时间受波特率、通讯模块、控制软件等多方面因素影响，此时间仅供参考。
2	读参数值命令	≤50ms	
3	写参数值命令	≤100ms	
4	PV 清零与微调命令	≤100ms	
5	恢复出厂设置命令	≤500ms	

表 21

附录 5——CRC16 计算方法

循环冗余校验(CRC)域为两个字节，包含一个二进制 16 位值。附加在报文后面的 CRC 的值由发送设备计算。接收设备在接收报文时重新计 CRC 的值，并将计算结果于实际接收到的 CRC 值相比较。如果两个值不相等，则为错误。

CRC 的计算，开始对一个 16 位寄存器预装全 1，然后将报文中的连续的 8 位子节对其进行后续的计算。只有字符中的 8 个数据位参与生成 CRC 的运算，起始位，停止位和校验位不参与 CRC 计算。

CRC 的生成过程中，每个 8 位字符与寄存器中的值异或。然后结果向最低有效位 (LSB)方向移动(Shift)1 位，而最高有效位(MSB)位置充零。然后提取并检查 LSB：如果 LSB 为 1，则寄存器中的值与一个固定的预置值异或；如果 LSB 为 0，则不进行异或操作。这个过程将重复直到执行完 8 次移位。完成最后一次（第 8 次）移位及相关操作后，下一个 8 位字节与寄存器的当前值异或，然后又同上面描述过的一样重复 8 次。当所有报文中子节都运算之后得到的寄存器中的最终值，就是 CRC.

生成 CRC 的过程:

1. 将一个 16 位寄存器装入十六进制 FFFF(全 1)，将之称作 CRC 寄存器。
2. 将报文的第一个 8 位字节与 16 位 CRC 寄存器的低字节异或，结果置于 CRC 寄存器。
3. 将 CRC 寄存器右移 1 位 (向 LSB 方向)，MSB 充零。提取并检测 LSB。
4. (如果 LSB 为 0): 重复步骤 3 (另一次移位)。

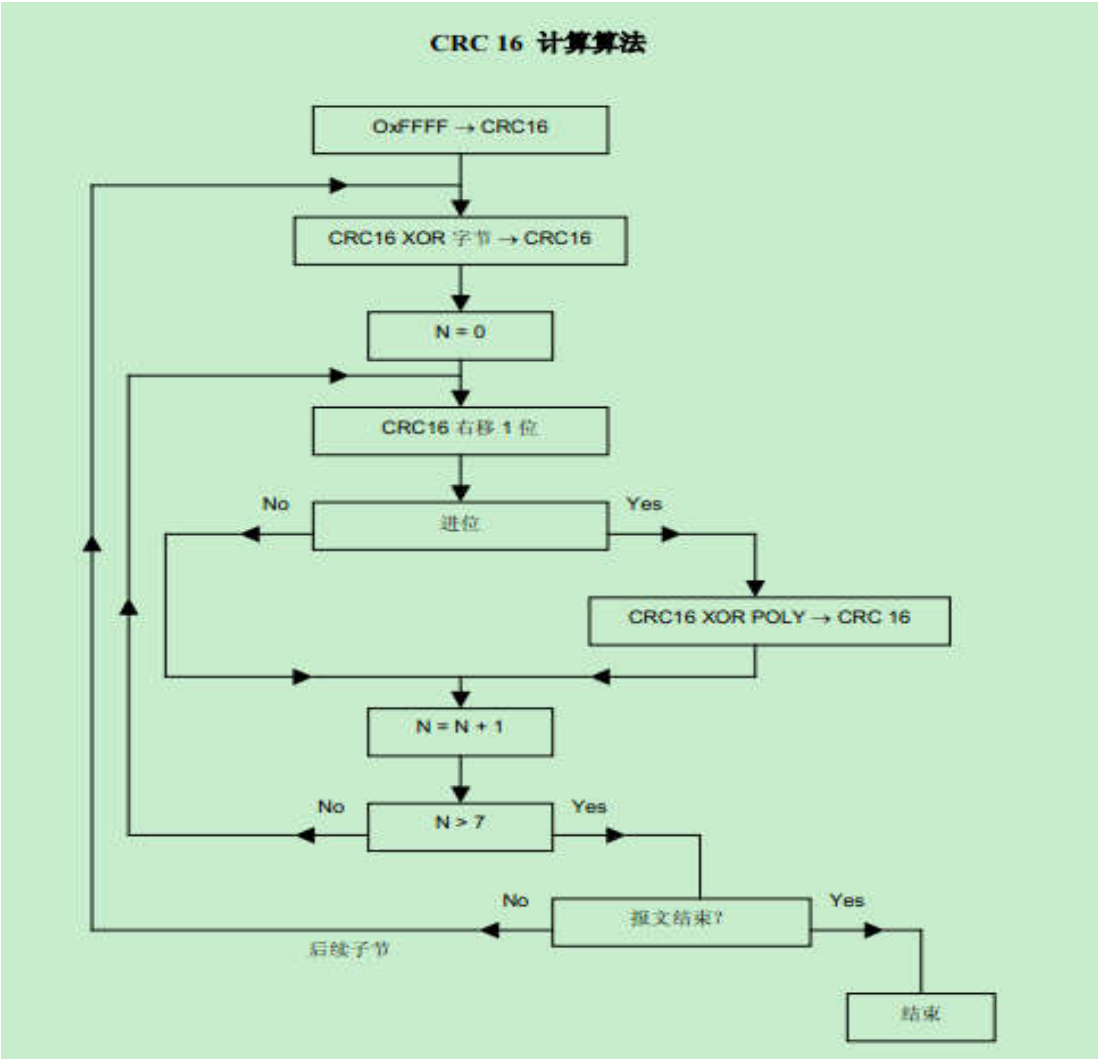
   (如果 LSB 为 1): 对 CRC 寄存器异或多项式值 0xA001 (1010 0000 0000 0001)。
5. 重复步骤 3 和 4，直到完成 8 次移位。当做完此操作后，将完成对 8 位字节的完整操作。
6. 对报文中的下一个字节重复步骤 2 到 5，继续此操作直至所有报文被处理完毕。
7. CRC 寄存器中的最终内容为 CRC 值.
8. 当放置 CRC 值于报文时，如下面描述的那样，高低字节必须交换。

当 16 位 CRC(2 个 8 位字节)在报文中传送时，低位字节首先发送，然后是高位节。

如果 CRC 值为十六进制 1241(0001 0010 0100 0001):

发送 CRC 时先发送 0x41，后发送 0x12。

CRC 16 计算算法：



XOR = 异或

N = 字节的信息位

POLY = CRC16 多项式计算 = 1010 0000 0000 0001

生成多项式 = 1 $+x_2+ x_{15}+ x_{16}$

在 CRC16 中，发送的第一个字节为低字节。

CRC 生成函数：

```
unsigned char *puchMsg;     //指向含有用于生成 CRC 的二进制数据报文缓冲区的指针
unsigned short usDataLen;   //报文缓冲区的字节数.

unsigned short CRC16 (puchMsg,usDataLen ); //函数以 unsigned short 类型返回 CRC
unsigned char *puchMsg ;                   // 用于计算  CRC 的报文
```

```
unsigned short usDataLen ;                              //报文中的字节数
{
    unsigned char uchCRCHi = 0xFF ;                     // CRC 的高字节初始化
    unsigned char uchCRCLo = 0xFF ;                     //CRC 的低字节初始化
    unsigned uIndex ;                                   //CRC 查询表索引

    while (usDataLen--)                                 //完成整个报文缓冲区
    {
        uIndex = uchCRCLo ^* puchMsgg++;                //计算 CRC
        uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex} ;
        uchCRCHi = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

高字节表

```
/* 高位字节的 CRC 值  */
static unsigned char auchCRCHi[] = {
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
};
```

低字节表

```
/* 低位字节的 CRC 值  */
static char auchCRCLo[] = {
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
};
```