成都远向电子科技有限公司产品说明书

产品名称：温度传感器 TS101U

全部资料下载地址：http://ask.zstel.com:8090

技术支持服务电话：028-64267900

技术支持专员企业 QQ：3183329475

官网网站：https://www.zstel.com/

硬件/软件技术定制热线：19150158475 张工

目录

一、产品概述

1.1 概述

温度传感器 TS101U 是我司推出的一款高性能温度测量产品，其测量范围-30℃~120℃（极限-40℃~125℃）。它能通过 USB 口直接输出测量到的温度数据，使用非常方便。广泛应用于工业生产、智能设备等场合。

1.2 性能特点

- USB 直接供电，自带 USB2.0 接口
- 支持 USB 数据主动输出
- 测量范围-30℃~120℃
- 支持标准 Modbus-RTU 通信协议
- 支持 USB 串口通信
- 高分辨率 0.1℃
- 四位数码管显示（可选）

1.3 技术参数

数据接口	通讯接口	USB
	波特率	1200～115200bps
	数据格式	8N1,8O1,8E1,8N2,8O2,8E2
	通讯协议	Modbus-RTU
测量范围	-30℃~120℃	
传输速率	0.1-30Hz	
精度	≤±0.5℃	
电源参数	电源规格	5V/0.5A
	功耗	5V-0.2W
工作环境	工作温度、湿度	-40℃～125℃，0%RH～95%RH
分辨率	0.1℃	
多连接	支持	

二、外观尺寸

2.1 产品外观

三、产品说明

3.1 设备接口

标准 USB2.0 接口

3.2 使用测试说明

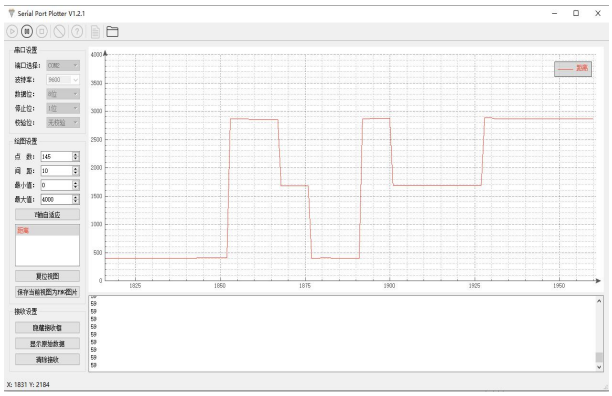
1.设备直接使用 USB 口接入电脑，可直接使用串口调试助手进行温度的读取和配置。

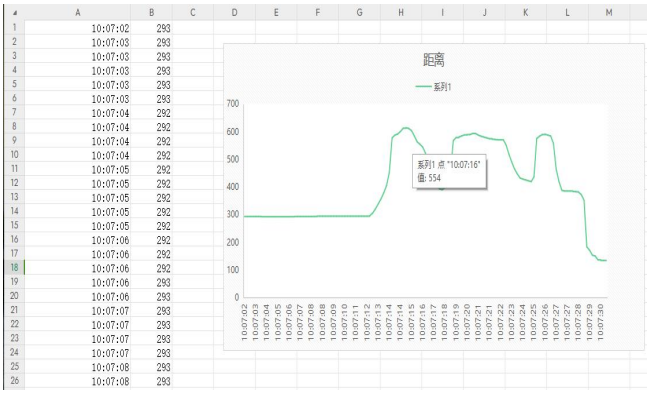
```
[2024-01-29 15:43:22.302]# SEND HEX>
01 03 00 00 00 01 84 0A

[2024-01-29 15:43:22.353]# RECV HEX>
01 03 02 00 AD 79 F9

[2024-01-29 15:43:22.571]# SEND HEX>
01 03 00 00 00 01 84 0A

[2024-01-29 15:43:22.619]# RECV HEX>
01 03 02 00 AC B8 39
```

2.设备支持标准 Modbus-RTU 协议，可读取设备 0x0000 地址获取温度数据，具体请查看第四章。

3.不带数码管显示传感器默认支持 AT 指令，可以配置脚本进行一些自定义操作，可主动输出温度值，例：AT+AUTO=300

以上例子便是将温度数据进行输出，以格式$24.00;自动上传，周期为 300ms，可结合 Serial Port Plotter 软件进行绘点，0 为不上传。可结合我司提供的软件查看折线图并保存测量数据为 CSV 表格。





注：AT 指令目前适配不带数码管显示的传感器探头。

3.4 传感器使用注意事项

1. 供电为直接使用 USB 进行供电，请严格按照要求进行供电，否则将很大概率烧毁探头。
2. 此探头为接触式探头，一般来说是探头表面温度。
3. 传感器默认串口为 USB 虚拟串口，通电 5 秒内以默认参数运行。
4. 传感器不具备防摔功能，所以要轻拿轻防以免传感器里面的测温探头摔坏，而影响使用。

四、ModbusRTU 通讯协议地址以及案例说明

4.1 通讯协议

本产品支持标准 Modbus RTU 从站协议，能够支持标准 Modbus RTU 组态软件，详细内容介绍参考本文第六章内容

4.2 寄存器地址

寄存器名称	寄存器地址	字节数	类型	备注
温度探头	0x0000（512）	2	整数	整数寄存器最高位为符号位，1 为负，0 为正，温度=寄存器值/10

4.3 Modbus RTU 功能码

功能码	操作	说明
03	读取温度寄存器值	读取温度寄存器值

详细讲解参照本文第六章内容

4.4 Modbus 通讯实例

（1）读取温度：

a. 用 03 功能码读取探头 1 整数温度：

发送：01 04 02 00 00 01 85 B2
接收：01 04 02 01 A5 78 3F
整数读出数值为 165（0x00A5）实际温度为 165/10=16.5 摄氏度

4.5 AT 指令集 （带 LED 显示的暂不支持 AT 指令集）

指令	备注
AT+READ=1	读取探头 1 的温度，2 为读取探头 2 的温度
AT+SET?	获取当前配置
AT+SET=C000,000	设置为摄氏度
AT+SET=F000,000	设置为华氏度
AT+AUTO=300	以格式$24.00;自动上传,周期为 300ms,可结合 Serial Port Plotter 软件进行绘点，0 为不上传
AT+SET=C-200	设置温度减少 20 度

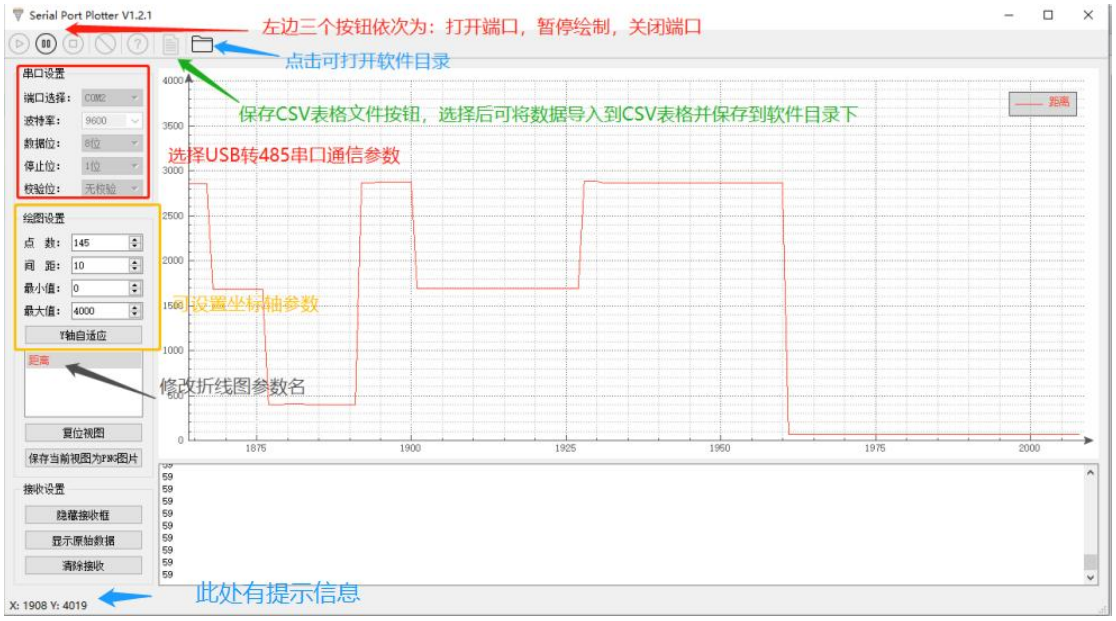
注：AT 指令配置成功返回 OK，重启设备生效，可直接拔出 USB 进行重启。

五、软件操作

5.1 温度数据可视化测试

我司提供一个可将距离数据形成折线图的软件,可实现简单的模块功能测试。

操作步骤如下图：



- 配置串口通信参数
- 勾选保存 csv 文件并打开串口
- 设置绘图设置，最小值为 0，最大值为 4000，点数和间距可自定义

注意：此时界面下方会出现当前的温度数据值。**若无上行数据请检查 USB-485 转换器是否正常工作，或尝试模块 485 AB 之间接 120 欧电阻。**

六、协议详解

地址域	功能码	数据	差错检验

Modbus 使用“big-Endian”（大端模式）表示地址和数据项，这就意味着当发射多个字节时，首先发送最高字节。

例如：寄存器地址为 0x0014，首先发送的是 0x00，然后才是 0x14。

一个正常的 Modbus 响应：响应功能码=请求功能码。

一个 Modbus 的异常响应：响应功能码=请求功能码+0x80，提供一个异常码来指示差错原因。

6.1 功能码描述

6.1.1 01 读线圈

可以使用此功能码读取继电器 DOx 的状态。

请求 PDU 详细说明了起始地址，即指定第一个线圈的地址和线圈数量，从零开始寻址线圈，因此寻址线圈 1-N 为 0-(N-1)。

响应 PDU 中 N 个字节的线圈状态的每一个 bit 位代表一个线圈的状态，状态 1=ON, 0=OFF。第一个字节的最低位 LSB 代表第 0 号线圈的状态（即起始地址指定的线圈号为 0 号线圈），其他线圈依次类推，一直到这个字节的最高位 MSB 为止，并且后续字节中都是由低到高代表连续的各线圈状态。

如果线圈数量不是 8 的倍数，将用零填充剩余最后数据字节中的剩余比特，字节数量域说明了数据的完整字节数。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x01
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 n-1)
CRC 校验	2 个字节	

注：线圈状态的字节数 N=线圈数量 n/8，如果余数不等于 0，则 N=n/8+1

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x81 （请求功能码+0x80）
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个读离散量 DO1 的实例

请求		响应	
地址	01	地址	01
功能码	01	功能码	01
起始地址高 H	00	字节数	01
起始地址低 L	14	DO1-DO4 状态	01
线圈数量高 H	00	CRC 校验高 H	90
线圈数量低 L	01	CRC 校验低 L	48
CRC 校验高 H	BD		
CRC 校验低 L	CE		

发送：010100140001BDCE　　RTU 响应：010101019048

DO1 的状态字节为 0D，二进制 00000001，DO1 是这个字节的 LSB(第 0 位)为 1 表示闭合，其他 DOx 是第(x-1)位为 0 表示断开，用 0 填充未使用位。

6.1.2　03 读保持寄存器/04 读输入寄存器

使用该功能码可以读取所有寄存器包括 AIx、DOx、DIx 的状态。

请求 PDU 详细说明了起始寄存器地址和寄存器数量，从 0 开始寻址寄存器，因此寻址寄存器 1-N 为 0-(N-1)。

响应报文中的寄存器数据每个寄存器有 2 个字节，对于每一个寄存器，第一个字节代表寄存器值的高位，第二个字节代表寄存器值的低位。字节数为寄存器数量乘以 2。对于 AI，一个通道占用 2 个寄存器，4 个字节的值使用浮点数表示，

对于 DOx,2 个字节的值 0000 代表继电器断开,0001 代表继电器闭合,对于 DIx,2 个字节的值 0000 代表开关量无输入,0001 代表有输入。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x03 或 04
起始地址	2 个字节	0x0000 至 0x0017
寄存器数量	2 个字节	n(1 至 N)
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x03 或 0x04
字节数	1 个字节	N=2*n
寄存器值	N 个字节	N=2*n, n 为寄存器数量
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x83 或 0x84 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个读模拟量输入 AI1 的实例

请求		响应	
地址	01	地址	01
功能码	03	功能码	03
起始地址高 H	00	字节数	04
起始地址低 L	00	AI1 值	4 个字节
寄存器数量高 H	00	CRC 校验高 H	
寄存器数量低 L	02	CRC 校验低 L	
CRC 校验高 H	C4		
CRC 校验低 L	0B		

发送：010300000002C40B RTU 响应:01030440199 99AD40F

6.1.3 05 写单个线圈

可以使用该功能码写单个继电器 DOx 为断开或闭合

请求数据域中的常量说明请求的 ON/OFF 状态，十六进制值 0xFF00 请求输出为 ON(闭合)，十六进制值 0x0000 请求输出为 OFF(断开)，其他所有值都是非法的，对输出不起作用，RTU 返回错误响应。

请求域中的输出地址规定了要写入线圈的地址。

正常响应是请求的应答，在写入线圈状态后返回这个正常响应。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x05
输出地址	2 个字节	0x0014 至 0x0015
输出值	2 个字节	0x0000 或 0xFF00
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x05
输出地址	2 个字节	0x0014 至 0x0015
输出值	2 个字节	0x0000 或 0xFF00
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x85 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求写线圈 DO1 为 ON(闭合)的实例

请求		响应	
地址	01	地址	01
功能码	05	功能码	05
输出地址高 H	00	输出地址高 H	00
输出地址低 L	14	输出地址低 L	14
输出值高 H	FF	输出值高 H	FF
输出值低 L	00	输出值低 L	00
CRC 校验高 H	CC	CRC 校验高 H	CC
CRC 校验低 L	3E	CRC 校验低 L	3E

发送：01050014FF00CC3E　　RTU 响应：01050014FF00CC3E

6.1.4　06 写单个寄存器

可以使用该功能码写单个继电器 DOx 为断开或闭合。

请求数据域中的寄存器值说明请求的 ON/OFF 状态，十六进制值 0001 请求输出为 ON(闭合)，十六进制值 0x0000 请求输出为 OFF(断开)。

请求域中的寄存器地址规定了要写入线圈的地址。

正常响应是请求的应答，在写入线圈状态后返回这个正常响应。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x06
寄存器地址	2 个字节	0x0014 至 0x0015
寄存器值	2 个字节	0x0000 至 0xFFFF
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x06
寄存器地址	2 个字节	0x0014 至 0x0015
寄存器值	2 个字节	0x0000 至 0xFFFF
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x86 (请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求写线圈 DO1 为 ON(闭合)的实例

请求		响应	
地址	01	地址	01
功能码	06	功能码	06
寄存器地址高 H	00	寄存器地址高 H	00
寄存器地址低 L	14	寄存器地址低 L	14
寄存器值高 H	00	寄存器值高 H	00
寄存器值低 L	01	寄存器值低 L	01
CRC 校验高 H	08	CRC 校验高 H	08
CRC 校验低 L	0E	CRC 校验低 L	0E

发送：0106001400010 80E　　RTU 响应：0106001400010 80E

6.1.5　0F 写多个线圈

可以使用此功能码写多个继电器 DOx 为断开或闭合。

请求 PDU 详细说明了起始地址，即指定第一个线圈的地址和线圈数量，从零开始寻址线圈，因此寻址线圈 1-N 为 0-(N-1)。

请求数据域中的内容说明了被请求的 ON/OFF 状态，域比特位中的逻辑“1”请求相应输出为 ON，域比特位中的逻辑“0”请求相应输出为 OFF。从数据域中第一个字节的 bit0 开始到 bit7，然后到第二个字节的 bit0，依次表示第一个线圈到第 n 个线圈的 ON/OFF 值。

正常响应返回功能码、起始地址和线圈数量。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x0F
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 N)
字节数	1 个字节	N=n/8,　　或 N=n/8+1
输出值	N 个字节	
CRC 校验	2 个字节	

注：线圈输出字节数 N=线圈数量 n/8，如果余数不等于 0，则 N=n/8+1

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x0F
起始地址	2 个字节	0x0014 至 0x0015
线圈数量	2 个字节	n(1 至 2)
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x8F　(请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个请求从线圈 DO1 开始写入 1 个线圈的实例

请求		响应	
地址	01	地址	01
功能码	0F	功能码	0F
起始地址高 H	00	起始地址高 H	00
起始地址低 L	14	起始地址低 L	14
线圈数量高 H	00	线圈数量高 H	00
线圈数量低 L	01	线圈数量低 L	01
字节数	01	CRC 校验高 H	D4
输出值	01	CRC 校验低 L	0F
CRC 校验高 H	DF		
CRC 校验低 L	54		

发送：010F0014000201012F51　　RTU 响应：010F00140001D40F

DO1 的输出值为 01，二进制 00000001，DO1 是这个字节的 LSB(第 0 位)为 0 表示断开，DOx 是第(x-1)位为 1 表示闭合，用 0 填充剩余未使用位。

6.1.6　10 写多个寄存器

使用该功能码可以写连续寄存器 DOx 的状态。

请求 PDU 详细说明了起始寄存器地址、寄存器数量、字节数和寄存器值，从

零开始寻址寄存器，因此寻址寄存器 1-N 为 0-(N-1)。

寄存器数据中每个寄存器有 2 个字节，对于每一个寄存器，第一个字节代表寄存器值的高位，第二个字节代表寄存器值的低位。字节数为寄存器数量乘以 2，2 个字节的值 0000 代表继电器断开，0001 代表继电器闭合。

正常响应返回功能码、起始地址和被写入寄存器的数量。

请求 PDU

地址	1 个字节	
功能码	1 个字节	0x10
起始地址	2 个字节	0x0014 至 0x0015
寄存器数量	2 个字节	n(1 至 N)
字节数	1 个字节	N=2*n
寄存器值	N 个字节	N=2*n,　　n 为寄存器数量
CRC 校验	2 个字节	

响应 PDU

地址	1 个字节	
功能码	1 个字节	0x10
起始地址	2 个字节	0x0014 至 0x0015
寄存器数量	2 个字节	n(1 至 2)
CRC 校验	2 个字节	

错误响应 PDU

地址	1 个字节	
功能码	1 个字节	0x90　(请求功能码+0x80)
异常码	1 个字节	0x01 或 0x02 或 0x03 或 0x04
CRC 校验	2 个字节	

这是一个控制继电器 DOx 的实例

请求		响应	
地址	01	地址	01
功能码	10	功能码	10
起始地址高 H	00	起始地址高 H	00
起始地址低 L	14	起始地址低 L	14
寄存器数量高 H	00	寄存器数量高 H	00
寄存器数量低 L	01	寄存器数量低 L	01
字节数	02	CRC 校验高 H	41
DO1 寄存器值高 H	00	CRC 校验低 L	CD
DO1 寄存器值高 L	01		
CRC 校验高 H	64		
CRC 校验低 L	84		

发送：01100014000102000164 84　　RTU 响应：01100014000141CD

D01 寄存器值为 0001 表示闭合

6.2 错误码描述

错误码含义：当 DTU 收到错误的 Modbus 指令时，会返回功能码为请求功能码+0x80，紧随着一个字节的错误码代表出错原因。

错误码 01：表示不支持的功能码，众山 DTU 支持上述 8 种功能码，除此之外的功能码都会返回错误码为 01 的错误。

错误码 02：表示起始地址不存在或者起始地址加上寄存器数量后的地址不存在。总的来说表示访问的寄存器不存在。

错误码 03：表示寄存器数量不符合规范或者寄存器值非法。

错误码 04：表示读写寄存器错误。

6.3 CRC 校验算法

CRC 即循环冗余校验码（Cyclic Redundancy Check ）：是数据通信领域中最常用的一种查错校验码，其特征是信息字段和校验字段的长度可以任意选定。循环冗余检查（CRC）是一种数据传输检错功能，对数据进行多项式计算，并将得到的结果附在帧的后面，接收设备也执行类似的算法，以保证数据传输的正确性和完整性。

ModbusRTU 的 CRC16 计算初值：0xFFFF

ModbusRTU 的 CRC16 计算多项式 0xA001（二进制:1010 0000 0000 0001）

附 CRC 校验算法代码：

```
uint16_t mb_crc( uint8_t* snd, uint16_t num )
{
        uint8_t CRC_Lb, CRC_Hb;
        uint8_t CRC_L, CRC_H;
        uint16_t crc;

        CRC_H = 0xFF;
        CRC_L = 0xFF;

        for ( uint16_t i = 0; i < num; i++ ) {
                CRC_L = CRC_L ^ snd[ i ];
                for ( uint16_t j = 0; j < 8; j++ ) {
                        CRC_Lb = CRC_L;
                        if ( ( CRC_L & 1 ) == 1 ) {
                                CRC_L = ( CRC_L - 1 ) / 2;
                                CRC_Lb = CRC_L;
                                CRC_Hb = CRC_H;
                                if ( ( CRC_H & 1 ) == 1 ) {
                                        CRC_L = CRC_L + 128;
```

```
                                    CRC_Lb  =  CRC_L;
                                    CRC_H   =  ( CRC_H  -  1 )  /  2;
                                    CRC_Hb  =  CRC_H;
                               }  else  {
                                    CRC_H   =  CRC_H  /  2;
                                    CRC_Hb  =  CRC_H;
                               }
                               CRC_L   =  CRC_L  ^  1;
                               CRC_Lb  =  CRC_L;
                               CRC_H   =  CRC_H  ^  0xA0;
                               CRC_Hb  =  CRC_H;
                          }  else  {
                               CRC_L   =  CRC_L  /  2;
                               CRC_Lb  =  CRC_L;
                               CRC_Hb  =  CRC_H;
                               if  (  ( CRC_H  &  1 )  ==  1 )  {
                                    CRC_L   =  CRC_L  +  128;
                                    CRC_Lb  =  CRC_L;
                                    CRC_H   =  ( CRC_H  -  1 )  /  2;
                                    CRC_Hb  =  CRC_H;
                               }  else  {
                                    CRC_H   =  CRC_H  /  2;
                                    CRC_Hb  =  CRC_H;
                               }
                          }
                     }
               }
               crc     =  CRC_L;
               crc  <<=  8;
               crc   |=  CRC_H;
               return  crc;
}
```