北京航空航天大學
BEIHANG UNIVERSITY

《单片机系统实验》报告三
——Modbus通信实验

北京航空航天大学研究生院

二〇二五年十一月

目　录

第一章　实验目的

通过本实验了解Modbus通信协议，能通过编程实现传感器或执行器的数据传输。

第二章　实验设备

STM32实验系统一套，个人笔记本（Windows操作系统，带USB接口）一台。

第三章　实验原理

了解STM32的UART通信，掌握对应Modbus协议的应用，并完成相关协议的编程，使传感器或执行器节点的数据通过Modbus协议传送给主控节点显示。

3.1　主控模块介绍

主控模块使用ARMCortex-M3内核的STM32系列处理器，作为物联魔盒中无线节点和传感器数据传输的桥梁，在传感器和无线节点之间进行数据交换，其具有的LCD屏可以方便的显示传感器的数据。主控模块平面展示图如图1所示。

图1　主控模块平面展示图

接口图如图2所示。

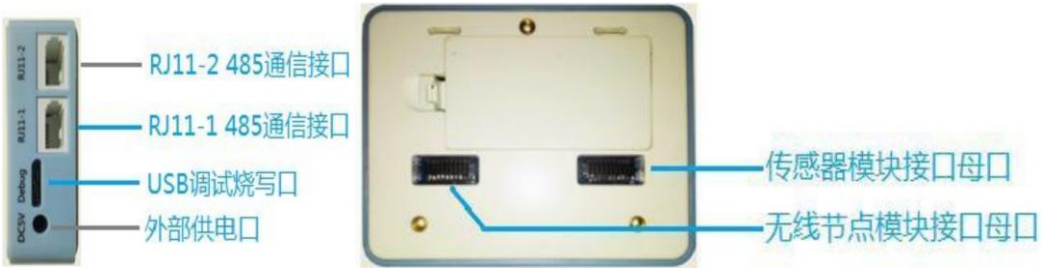

图2　主控模块接口图

传感器模块接口用来插接传感器模块，而无线节点模块接口用来插接无线节点模块，此次实验中没有用到无线节点。感知执行模块与主控模块的连接如图3所示。

图3　主控模块与感知执行模块、无线模块连接示意图

3.2　Modbus协议介绍

Modbus通信协议被广泛应用于主从模式的网络中，**硬件基础可以是RS232，RS485等多种形式**。在本实验中，硬件采用了RS485。RS485是一种基于差分信号的硬件协议（请查阅相关资料），两条线一个信道进行数据通信，是半双工的通信模式。因此在使用时，需要在硬件上有一个收发数据的控制端。

对本次实验的从节点（传感器/执行器节点）来说，编程方面还是用串口进行数据的收发，但会有一个**控制口线（图中485_CTRL_P1）进行数据收发的控制**。

	A	B	C	D	E	F	G
1	传感器	Status	RS485-R	RS485-T	User1	ADC	485_CTRL
2	触摸传感器	GPIOB_4	GPIOB_5	GPIOB_6	GPIOB_7		PA1
3	光敏传感器	GPIOB_9	GPIOB_8	GPIOB_5	GPIOB_4	AD为PA4	PB0
4	温湿度传感器	GPIOB_4	GPIOB_5	GPIOB_6	GPIOB_7		PA1
5	空气传感器	GPIOB_9	GPIOB_8	GPIOB_5	GPIOB_4	AD为PA4	PB0
6	步进电机驱动器	GPIOB_4	GPIOB_5	GPIOB_6	GPIOB_7		PA1
7	人体红外传感器	GPIOB_4	GPIOB_5	GPIOB_6	GPIOB_7		PA1
8	风扇						PB0
9	多彩LED	GPIOB_9	GPIOB_8	GPIOB_5	GPIOB_4		PB0
10							

图4　控制线接口

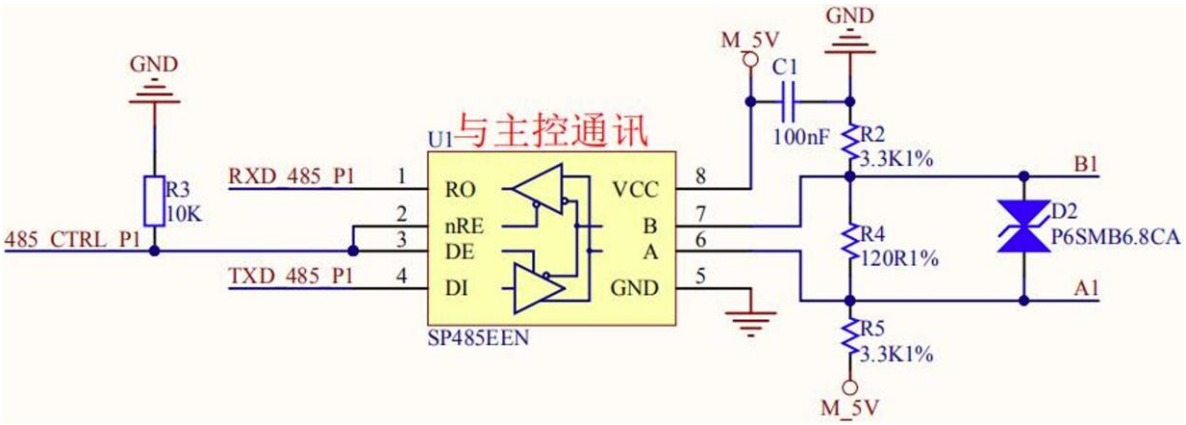

图5　硬件电路部分示意图

传感器节点通过串口与主控模块进行通信，二者的通信符合Modbus通信协议，其中主控模块为主机，终端节点为从机，协议规定主机以一定的数据格式向从机发送命令数据时，从机也必须以规定的数据格式回复从机以应答数据。在本实验中我们采用Modbus通信协议中的RTU（远程终端单元）格式。

当通讯命令发送至仪器时，符合相应地址码的设备接到讯命令，并除去地址码，读取信息，如果没有出错，则执行相应的任务；然后把执行结果返送给发送者。返送的信息中包括地址码、执行动作的功能码、执行动作后结果的数据以及错误校验码。如果出错就不发送任何信息。

3.3 命令帧信息结构

地址码	功能码	数据区	CRC 校验码
8 位（1 字节）	8 位（1 字节）	N*8 位（N*1 字	16 位（2 字节）

图6 命令帧信息结构

（1）地址码：地址码是信息帧的第一字节(8 位)，从 0 到 255。这个字节表明由用户设置地址的从机将接收由主机发送来的信息。每个从机都必须有唯一的地址码，并且只有符合地址码的从机才能响应回送。当从机回送信息时，相当的地址码表明该信息来自于何处。

（2）功能码：主机发送的功能码告诉从机执行什么任务。

（3）数据区：数据区包含需要从机执行什么动作或由从机采集的返送信息。这些信息可以是数值、参考地址等等。例如，功能码告诉从机读取寄存器的值，则数据区必需包含要读取寄存器的起始地址及读取长度。对于不同的从机，地址和数据信息都不相同。

（4）错误校验码：主机或从机可用校验码进行判别接收信息是否出错。

3.4 功能码

功能码	名称	功能
03（0x03）	读取保持寄存器	在一个或多个保持寄存器中取得当前的二进制值
06（0x06）	写单个寄存器	把具体二进值装入一个保持寄存器
23（0x17）	读/写多个保存寄存器	这个功能码实现了一个读操作和一个写操作的组合

图7 功能码

（1）0x03：读取保持寄存器；

（2）06(0x06)写单个保持寄存器；

（3）16(0x10)写多个寄存器；

（4）23(0x17)读/写多个寄存器。

3.5 地址码

地址码是信息帧的第一字节(8位)，从0到255。这个字节表明由用户设置地址的从机将接收由主机发送来的信息。每个从机都必须有唯一的地址码，并且只有符合地址码的从机才能响应回送。当从机回送信息时，相当的地址码表明该信息来自于何处。

传感器/执行器	地址码	感知量/执行梁	长度（Byte）	数据组成结构
温湿度传感器	20H	温度、湿度	4	温度 2 字节：高 8 位，温度整数部分；低 8 位，小数部分 湿度 2 字节：高 8 位，湿度整数部分；低 8 位，小数部分
光敏传感器	24H	光照强度	2	0-65535
空气质量传感器	25H	浓度	2	10-5000ppm
触摸传感器	64H	触碰	2	高 8 位：00 低 8 位：1（检测到）0（未检测到）
步进电机	96H		2	高 8 位：00 低 8 位：执行状态
角度伺服电机	98H	旋转角度	2	高 8 位：00 低 8 位：0-180 代表舵机旋转的角度

图8 地址码

3.6 寄存器分配地址

本实验系统制定的Modbus协议，对系统的寄存器有很全面的描述，例如产品的ID，波特率的设置等都有定义，这部分内容我们在实际实验中没有用到，因此不做描述。在本部分，只给出了实验用传感器和执行器的数据存放的寄存器地址。

	2000H	2001H	2100H	2200H	2201
温湿度传感器	温度	湿度	—	—	—
光敏传感器	光照强度（0-65535）	—	—	—	—
空气质量传感器	检测浓度（10—5000ppm）	—	—	—	—
触摸传感器	—	—	1：检测到 0：未检测到	—	—
步进电机	—	—	—	1：正转 2：反转 3：停	电机转速（m/s）
角度伺服电机	—	—	—	旋转角度	—

图9 寄存器分配地址

第四章　实验内容与步骤

4.1　实验内容

理解掌握相关Modbus通信协议， 编程实现Modbus部分协议，实现传感器/执行器与主控模块之间的数据传输。（8学时 ）

（1）USART2正确配置（时钟，波特率，中断等）；

（2）知道自己的节点地址；

（3）侦听数据（只获取有效数据）；

（4）作出响应。

4.2　实验步骤

（1）接线：使用一根蓝色数据线将电脑的USB接口与ST-LINK底部的USB-Debug-调试下载口连接；再使用一根白色数据线，将ST-LINK顶部的 Debug-USB调试下载口连接至感知执行模块底部的USB3.0调试烧写口；使用同样的蓝色连接线将电脑的USB接口与ST-LINK底部的USB-485-USB转RS-485接口相连，再使用另一根白色连接线，将ST-LINK顶部的RS-485-RJ11-485通信接口与感知执行模块底部的RJ11-485通信接口相连。

（2）引用头文件。

```
#include "servo.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_tim.h"
```

（3）宏定义：定义了舵机Modbus地址、指令寄存器地址、接收缓冲区最大长度，以及舵机可转动角度范围。

```
// 舵机与通信参数定义
#define SERVO_ADDR 0x98       // 舵机地址码
#define SERVO_REG_ADDR 0x2200 // 舵机指令寄存器
#define MODBUS_RX_MAX_LEN 8   // 最大接收长度
#define ANGLE_MIN 0           // 最小角度
#define ANGLE_MAX 180         // 最大角度
```

（4）全局变量：全局变量包括接收缓冲区、接收计数器以及舵机默认角度。

```
// 全局变量
uint8_t modbus_rx_buf[MODBUS_RX_MAX_LEN]; // 接收缓冲区
volatile uint8_t rx_cnt = 0;              // 接收计数（加 volatile）
uint8_t target_angle = 90;                // 目标角度(默认 90 度)
```

（5）延时函数：循环延时用于RS485切换方向或PWM信号等待。

```
// 函数声明
void MODBUS_Send_Reply(uint8_t func_code);
void Clear_Modbus_Buffer(void);

// Delay
```

```c
void Delay(u32 count)
{
    u32 i = 0;
    for (; i < count; i++)
        ;
    // 通过循环来实现延时
}
```

（6）CRC16校验：根据Modbus RTU协议计算CRC16，保证指令传输的完整性，避免舵机收到错误命令。

```c
// CRC16 校验计算(MODBUS-RTU 协议)
uint16_t CRC16_Calculate(uint8_t *buf, uint8_t len)
{
    uint16_t crc = 0xFFFF;
    uint8_t i, j;
    for (i = 0; i < len; i++)
    {
        crc ^= buf[i];
        for (j = 0; j < 8; j++)
        {
            if (crc & 0x0001)
            {
                crc >>= 1;
                crc ^= 0xA001;
            }
            else
            {
                crc >>= 1;
            }
        }
    }
    return crc;
}
```

（7）RS485接口初始化：配置PA1为RS485方向控制引脚，高电平发送，低电平接收。

```c
// RS485 初始化(控制引脚 PA1)
void RS485_IO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;        // 485 方向控制引脚 PA1
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // 推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_ResetBits(GPIOA, GPIO_Pin_1); // 初始化为接收模式(低电平)
}
```

（8）USART2初始化：配置PA2为TX、PA3为RX，波特率9600，8位数据位、1位停止位、无校验，开启接收中断。

```c
// USART2 初始化(9600 8N1)
void USART2_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
```

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE)
;

    // 配置 TX 引脚(PA2)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // 配置 RX 引脚(PA3)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    // 配置 USART2
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_
None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART2, &USART_InitStructure);

    // 配置接收中断
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    USART_Cmd(USART2, ENABLE);
}
```

（9）串口接收与缓冲区管理：提供缓冲区清空函数，并在USART中断中接收数据，同时处理IDLE中断用于帧结束判断。

```
// 清除接收缓冲区
void Clear_Modbus_Buffer(void)
{
    uint8_t i; // 声明循环变量在函数开始处
    rx_cnt = 0;
    for (i = 0; i < MODBUS_RX_MAX_LEN; i++)
    {
        modbus_rx_buf[i] = 0;
    }
}

uint8_t current_rx_cnt;
// USART2 中断服务函数(接收数据)
void USART2_IRQHandler(void)
{
    // uint8_t rx_data; // 临时存储接收的数据
    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        if (rx_cnt < MODBUS_RX_MAX_LEN)
        {
            modbus_rx_buf[rx_cnt++] = USART_ReceiveData(USART2);
        }
        else
        {
```

```
            Clear_Modbus_Buffer(); // 缓冲区满则清空并等待下一帧
        }
        USART_ClearITPendingBit(USART2, USART_IT_RXNE); // 清除中断标志
    }
    else if (USART_GetITStatus(USART2, USART_IT_IDLE) != RESET)
    {
        current_rx_cnt = rx_cnt;
        rx_cnt = 0;
        USART_ClearITPendingBit(USART2, USART_IT_IDLE); // 清除中断标志
    }
    USART_ReceiveData(USART2);
}
```

（10）Modbus指令解析：实现了地址验证、数据长度校验、CRC校验，并解析功能码，实现读写寄存器指令处理。

```
// MODBUS 指令解析
void MODBUS_Parse_Command(void)
{
    // 所有局部变量声明放在函数开头（符合 C89 标准）
    uint16_t received_crc;
    uint16_t calculated_crc;
    uint8_t func_code;
    uint16_t reg_addr;
    // 校验地址
    if (modbus_rx_buf[0] != SERVO_ADDR)
    {
        Clear_Modbus_Buffer();
        return;
    }
    // 至少需要地址+功能码+两个字节寄存器地址+两个字节 CRC 才能校验
    if (current_rx_cnt < 6)
    {
        return; // 不清空缓冲区，等待更多数据
    }
    // 校验 CRC
    received_crc = (modbus_rx_buf[current_rx_cnt - 1] << 8) | modbus_rx_buf[cu
rrent_rx_cnt - 2];
    calculated_crc = CRC16_Calculate(modbus_rx_buf, current_rx_cnt - 2);
    if (received_crc != calculated_crc)
    {
        Clear_Modbus_Buffer();
        return;
    }
    func_code = modbus_rx_buf[1];
    reg_addr = (modbus_rx_buf[2] << 8) | modbus_rx_buf[3];
    // 处理写寄存器指令(0x06)
    if (func_code == 0x06 && reg_addr == SERVO_REG_ADDR && current_rx_cnt >= 8
)
    {
        target_angle = modbus_rx_buf[5];
        // 限制角度范围
        if (target_angle < ANGLE_MIN)
            target_angle = ANGLE_MIN;
        if (target_angle > ANGLE_MAX)
            target_angle = ANGLE_MAX;
        Servo_Move(target_angle);
        MODBUS_Send_Reply(0x06);
        Clear_Modbus_Buffer();
    }
    // 处理读寄存器指令(0x03)
    else if (func_code == 0x03 && reg_addr == SERVO_REG_ADDR && current_rx_cnt
 >= 6)
```

```
    {
        MODBUS_Send_Reply(0x03);
        Clear_Modbus_Buffer();
    }
    else
    {
        // 如果数据不完整，不清空缓冲区，等待更多数据
        return;
    }
}
```

（11）Modbus应答发送：根据指令类型生成应答帧，切换RS485收发方向并发送数据，确保通信安全可靠。

```
// 发送 MODBUS 应答
void MODBUS_Send_Reply(uint8_t func_code)
{
    uint8_t tx_buf[8];
    uint8_t tx_len = 0;
    uint16_t crc;
    uint8_t i;

    tx_buf[0] = SERVO_ADDR;
    tx_buf[1] = func_code;

    if (func_code == 0x03)
    {
        // 读指令应答: 地址+功能码+数据长度+数据(2 字节)+CRC
        tx_buf[2] = 0x02;         // 数据长度
        tx_buf[3] = 0x00;         // 角度高 8 位
        tx_buf[4] = target_angle; // 角度低 8 位
        tx_len = 5;
    }
    else if (func_code == 0x06)
    {
        // 写指令应答: 地址+功能码+寄存器地址+数据+CRC
        tx_buf[2] = (SERVO_REG_ADDR >> 8) & 0xFF; // 寄存器高 8 位
        tx_buf[3] = SERVO_REG_ADDR & 0xFF;        // 寄存器低 8 位
        tx_buf[4] = 0x00;                         // 数据高 8 位
        tx_buf[5] = target_angle;                 // 数据低 8 位
        tx_len = 6;
    }

    // 计算 CRC
    crc = CRC16_Calculate(tx_buf, tx_len);
    tx_buf[tx_len++] = crc & 0xFF;        // CRC 低 8 位
    tx_buf[tx_len++] = (crc >> 8) & 0xFF; // CRC 高 8 位

    // 关闭接收中断防止冲突
    USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);

    // 切换为发送模式
    GPIO_SetBits(GPIOA, GPIO_Pin_1);
    Delay(100); // 确保方向切换完成

    // 发送数据
    for (i = 0; i < tx_len; i++)
    {
        while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET)
            ;
        USART_SendData(USART2, tx_buf[i]);
    }
```

```
    // 等待发送完成
    while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET)
        ;
    Delay(100);
    GPIO_ResetBits(GPIOA, GPIO_Pin_1); // 切换回接收模式

    // 重新开启接收中断
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}
```

（12）主函数：完成外设初始化，将舵机置于初始角度90°，主循环仅根据接收到的数据触发指令解析，而不直接处理串口中断数据。

```
int main(void)
{
    // 初始化外设
    RS485_IO_Init();
    USART2_Init();
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
    USART_ITConfig(USART2, USART_IT_IDLE, ENABLE);
    Servo_TIM3_PWM_Init();

    Servo_Move(90); // 初始位置

    GPIO_ResetBits(GPIOA, GPIO_Pin_1);

    while(1)
    {
    // 只有在接收到足够数据时才处理
    if (current_rx_cnt == 8) // 至少需要地址+功能码+寄存器地址+部分 CRC
    {
        MODBUS_Parse_Command();
        current_rx_cnt = 0;
    }
}
}
```

（13）串口调试助手：设置波特率为与程序一致的9600，选择正确端口，选择HEX发送和HEX显示模式。

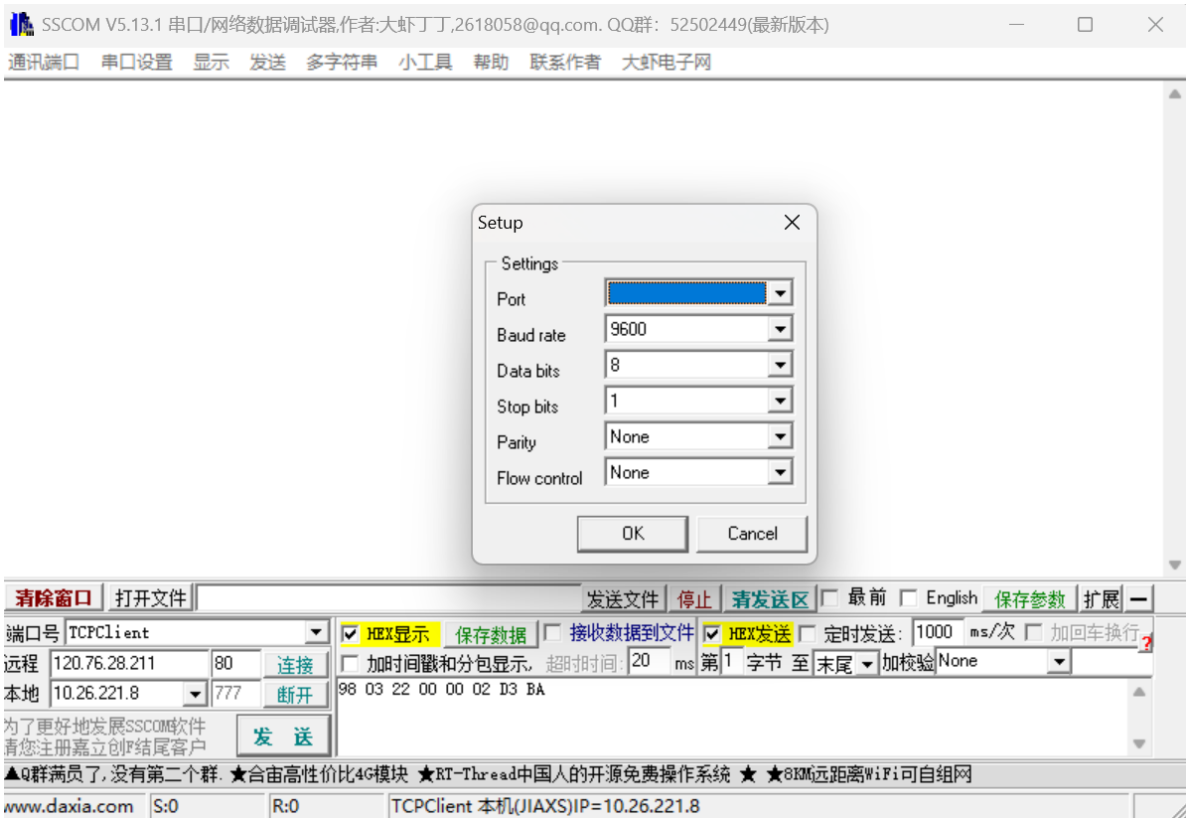

图10　串口调试助手

（14）编译与下载程序：点击工具栏依次执行Translate、Build、Download，烧录完成后，重新上电（与前两个实验不同），将主机连接至舵机并打开主机电源，主机会开始持续发送数据，串口调试助手中也能看到这些数据。一段时间后，主机即可正确识别出所连接的执行器为舵机，并检测到此时舵机的角度为 90°。