CM2003 ModbusRTU 采集热电偶传感器

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码介绍如何使用 CM2003 的 USCI 接口连接 NADC24 采集热电偶传感器的温度信号。
BSP 版本	CM2003_Series_BSP_CMSIS_V3.00.000
开发平台	CM2003_ModbusRTU_NADC24_V0.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例代码介绍了 CM2003 通过 USCI 接口读取 NADC24 的差分通道，用以采集外接的热电偶传感器的温度信号。被读取的温度信号使用基于 RS485 接口的 Modbus 协议传送至 USB to RS485 adaptor, USB to RS485 adaptor 再传至 PC 以便接收分析使用。

Modbus 协议使用了开源 FreeModbus 协议栈。FreeModbus 因其开源、轻量和易移植的特点，在嵌入式 Modbus 设备开发中被广泛采用。开发者可以根据项目需求进行定制和优化。

更多关于 FreeModbus 的介绍请查看其官网介绍：

https://www.embedded-experts.at/en/freemodbus/about/

1.1 CM2003 介绍

本例程需要 CM2003 的 3 种基本功能，UART、SPI、TIMER：

- UART：接口通过 RS485 负责实现 Modbus 协议的数据的收发。
- SPI：接口负责读写 NADC24 的数据，因 CM2003 没有标准 SPI 接口，使用了 USCI 接口的 SPI 功能来实现。NADC24 有 8-bit 和 24-bit 数据模式，但 USCI 只能支持 8-bit 和 16-bit 两种模式，因此例程 USCI 统一按照 8-bit 模式来进行数据的读写。
- TIMER：定时器 0 用于判断通讯超时，Modbus 协议要求每一帧数据间隔要有 ≥ 3.5 个位信号的时间。定时器 1 用于产生秒计数。

1.2 Modbus 协议简介

Modbus 使用主从架构，且有三种协议类型：

- Modbus RTU（基于串行通信，如 RS-485/RS-232，二进制编码，高效紧凑）。
- Modbus ASCII（基于串行通信，ASCII 字符编码，可读性强但效率低）。
- Modbus TCP/IP（基于以太网，使用 TCP 端口，适用于现代工业网络）。

协议的主要内容为：设备地址 + 指令 + 寄存器地址 + 数据 + CRC 校验

其中 8 种指令见表 1-1：

功能码(十进制)	操作	数据类型
1	读取线圈状态	1-bit（开关量）
2	读取离散输入	1-bit（只读）
3	读取保持寄存器	16-bit（可读写）
4	读取输入寄存器	16-bit（只读）
5	写入单个线圈	1-bit（开关量）
6	写入单个保持寄存器	16-bit（可读写）
15	写入多个线圈	1-bit（批量操作）
16	写入多个保持寄存器	16-bit（批量操作）

表 1-1 Modbus 指令表

本例程中使用了 Modbus 协议中 4 种功能码：3、4、6、16。

例程中定义的 Modbus 通讯格式见表 1-2：

通讯参数	参数值
Address	1
Baud_rate	115200
Data_bits	8
Stop_bits	1
Parity	无
Flow_control	无

表 1-2 例程通讯格式

寄存器定义参见表 1-3：

寄存器地址(十进制)	寄存器类型	定义	数据格式
1000	输入寄存器（只读）	硬件版本	16-bit HEX
1001		软件版本	16-bit HEX

寄存器地址 (十进制)	寄存器类型	定义	数据格式
1002		秒计数	1 秒
1003		输入口状态	0/1
1004		热电偶温度	0.1 ℃
1005		NADC24 温度	0.1 ℃
2000	保持寄存器（可读写）	LED 状态	0/1
2001		IO 输出状态	0/1
2002		NADC24 通道设置	同 NADC 通道寄存器
2003		MCU 工作模式	未启用

表 1-3 寄存器数据

更多 Modbus 的介绍请查看 Modbus 组织的协议文档：

https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

1.3 热电偶温度采集原理

当两种不同的金属导体（或半导体）两端连接形成闭合回路时，若两个连接点（称为热端和冷端）存在温度差（ΔT），回路中会产生电动势（电压），这种现象称为塞贝克效应（Seebeck Effect）。产生的电压称为热电势，其大小与温度差成正比，关系式为：

$$V = \alpha \cdot \Delta T$$

其中：

- V：热电势（单位：mV）。
- α：塞贝克系数（材料特性，单位：μV/°C）。
- ΔT：热端与冷端的温度差（单位：°C）。

热电偶测量的基本结构：

- 热端（测量端）：暴露在待测温度环境中，温度 T_1。
- 冷端（参考端）：保持已知恒定温度（如 0 °C）或通过补偿电路修正，温度 T_2。
- 导线材料：两种不同的金属导体。

热电偶通过测量热电势 V 来推算热端温度 T_1，但需已知冷端温度 T_2。

若热电偶冷端温度不为 0 °C，而是 T_2，则热电势 V (T_1 , T_2) 与标准分度表 (冷端为 0 °C) 的关系为：

$$V(T_1, 0) = V(T_1, T_2) + V(T_2, 0)$$

1.3.1 热电偶分类

热电偶根据其导体材料的不同，有多种规格可供选择

- K 型热电偶：由镍铬-镍硅材料构成

  适用于 -200 ℃至 1300 ℃ 的温度范围，广泛应用于工业领域。

- J 型热电偶：由铁-康铜材料构成

  适用于 -200 ℃至 750 ℃ 的温度范围，适合在氧化性气体环境中使用。

- E 型热电偶：由镍铬-铜镍材料构成

  适用于 -200 ℃至 900 ℃ 的温度范围，具有较高的热电动势。

- T 型热电偶：由铜-康铜材料构成

  适用于 -200 ℃至 350 ℃ 的温度范围，适合低温测量。

- N 型热电偶：由镍铬硅-镍硅镁材料构成

  适用于 -200 ℃至 1300 ℃ 的温度范围，具有良好的抗氧化性。

例程使用了通用性较广的 K 型热电偶，并且符合 ITS-90 国际温标。关于 ITS-90 的更多信息请查看官方定义：

https://www.nist.gov/system/files/documents/pml/div685/grp01/ITS-90_metrologia.pdf

K 型温度和电势的对应关系：当 0 ℃ 对应 0 mV 时

-270 ℃	<===	0 ℃	===>	1372 ℃
-6.458 mV	<===	0 mV	===>	54.886 mV

详细的温度电压对应关系可查看链接文件：

https://www.omega.co.uk/temperature/pdf/Type_K_Thermocouple_Reference_Table.pdf

1.3.2 热电偶温度计算方法

热电偶的温度计算方法有 2 种，查表法和计算法。两种计算方法均被 ITS-90 认可。

- 查表法：通常用在處理器主頻不高的情況下，范例代码选用了查表法进行温度计算。
- 计算法：通过多项式的方程可计算出温度。

1.3.3 例程测热电偶的步骤：

1. 读取 NADC24 内部温度标定值；
2. 根据标定值计算 NADC24 内部温度传感器温度（冷端温度）；
3. 冷端温度转换成冷端电压；
4. 测量热电偶热端电压；
5. 通过热端电压和冷端电压计算绝对温度。

1.4 NADC24 介绍

热电偶的电势变化很小，常规的 ADC 无法进行精准的采集，需要分辨率高，差分通道才能精确的采集热电偶的电池变化。例程中使用的 NADC24 有 3 组差分通道，并且内部有温度传感器可作为冷端的温度采集；CH6 和 CH7 组成一组差分通道用来采集热端温度。

NADC24 每采样一次产生一个中断信号，可以设置为连续采样或者单次采样。例程中使用增益为 32 倍，采样率为 32 kbps 进行连续采样。

2. 代码介绍

代码主要分成 4 部分：

- 程序主框架：负责初始化，数据处理。
- FreeModbus 协议栈：负责 Modbus 协议通讯参数设置，数据收发。
- NADC24 电压测量：包含 IC 的初始化，标定程序，数据采集。
- 热电偶温度计算：热电偶温度的计算。

2.1 程序主框架

主程序开始对系统进行初始化，包括 CM2003 内部接口的初始化，Modbus 协议的初始化在 *SYS_Init()* 函数内，NADC24 的初始化，Modbus 协议寄存器的初始化。主循环内 *eMBPoll()* 负责 Modbus 协议的解析，*NADC24_Loop()* 在检测转换结束时由中断标志位触发运行。

```
/*---------------------------------------------------------------*/
/* MAIN function                                                 */
/*---------------------------------------------------------------*/
int main(void)
{
    /* Init System, IP clock and multi-function I/O */
    SYS_Init();
    LED_PIN = 0;
    NADC24_Init();// Input: N_TP, P_TP ,VREF=2.4V

     RegInput[HW_VER_REG] =    HW_VERSION;
     RegInput[SW_VER_REG] =    SW_VERSION;

    /* Set PB multi-function pin for EINT1 (PB.4) */
    SYS->GPB_MFPL = (SYS->GPB_MFPL & ~( SYS_GPB_MFPL_PB4MFP_Msk ) |
   ( SYS_GPB_MFPL_PB4MFP_INT1 ));
    GPIO_EnableInt(PB, 4, GPIO_INT_FALLING);//NADC24 sampling completion interrupt.
    NVIC_EnableIRQ(EINT1_IRQn);

    TIME3_Init();

    while (1)
    {
        /*eMBPoll for modbus event process*/
        (void)eMBPoll();
        NADC24_Loop(); //NADC24 main loop.

        if(PB13 == 1)
            RegInput[MCU_DIN_REG] = 1;
        else
            RegInput[MCU_DIN_REG] = 0;
```

```
    };
}
```

2.2 FreeModbus 协议栈

FreeModbus 协议栈移植完成后只需要关注 *user_modbus.c* 文件中 4 种寄存器类型数据的处理，例程中仅使用了输入寄存器和保持寄存器 2 种寄存器。输入寄存器只向外部发送数据，保持寄存器即可以写入也能发送，如果用户需要非易失，就在在保持寄存器中加入 Flash 的写入功能。

```
/**********************************************************************************
 *  @brief        Read input Registers
 *  @param[in]    pucRegBuffer: Data pointer.
 *                   usAddress: Register address.
 *                     usNRegs: Number of register operations.
 *  @return       Modbus operational Status.
 *  @details      Only send out data int input Registers.
**********************************************************************************/
eMBErrorCode eMBRegInputCB(UCHAR *pucRegBuffer, USHORT usAddress, USHORT usNRegs)
{
    eMBErrorCode    eStatus = MB_ENOERR;
    int             iRegIndex;

    usAddress--;

    if ((usAddress >= REG_INPUT_START)
            && (usAddress + usNRegs <= (REG_INPUT_START + REG_INPUT_NREGS)))
    {
        iRegIndex = (int)(usAddress - usRegInputStart );

        while (usNRegs > 0)
        {
            *pucRegBuffer++ = (unsigned char)(RegInput[iRegIndex] >> 8);
            *pucRegBuffer++ = (unsigned char)(RegInput[iRegIndex] & 0xFF);
            iRegIndex++;
            usNRegs--;
        }
    }
    else
    {
        eStatus = MB_ENOREG;
    }

    return eStatus;
}

/**********************************************************************************
 *  @brief        Read & write Holding Registers
 *  @param[in]    pucRegBuffer: Data pointer.
 *                   usAddress: egister address.
 *                   usNRegs: Number of register operations.
 *                      eMod: MB_REG_READ or MB_REG_WRITE
 *  @return     Modbus operational Status.
 *  @details    The register operation can be placed in Hold_Reg_Refresh().
```

```
**********************************************************************************/
eMBErrorCode eMBRegHoldingCB(UCHAR *pucRegBuffer, USHORT usAddress, USHORT usNRegs,
eMBRegisterMode eMode)
{

    eMBErrorCode    eStatus = MB_ENOERR;
    int             iRegIndex;

    usAddress--;

    if ((usAddress >= REG_HOLD_START)
            && (usAddress + usNRegs <= (REG_HOLD_START + REG_HOLD_NREGS)))
    {    iRegIndex = (int)(usAddress - usRegHoldStart);

         if(eMode == MB_REG_READ)
         {
              while (usNRegs > 0)
              {
              *pucRegBuffer++ = (unsigned char)(RegHold[iRegIndex] >> 8);
              *pucRegBuffer++ = (unsigned char)(RegHold[iRegIndex] & 0xFF);
              iRegIndex++;
              usNRegs--;
              }
         }
         else
         {
              while (usNRegs > 0)
              {
              RegHold[iRegIndex]  = (*pucRegBuffer++)<<8;
              RegHold[iRegIndex]  |= *pucRegBuffer++ ;
              iRegIndex++;
              usNRegs--;
              }
         }
    }
    else
    {
        eStatus = MB_ENOREG;
    }
    return eStatus;
}
```

2.3 NADC24 电压测量

NADC24 在初始化的时候已经做好了测量电路的标定。内部温度传感器在系统启动的时候需进行第一次采样（冷端采样），采样完成后将通道再切换到热电偶的采样通道持续采样（热端采样）。2 种采样均通过 16 次平均采样，32 kbps 的采样率 16 次采样结束后将温度计算后放入 Modbus 相应的输入寄存器内。需要注意的是进行 16 次平均后，实际温度的更新速率大约为 0.5 秒。

```
/**********************************************************************************
 *  @brief      NADC24 main loop.
```

```
 *  @param[in]  NULL
 *  @return     NULL
 *  @details    Put it in the main loop of the program.
It needs to run to completion before the next sampling completion flag is set.
*****************************************************************************/
void NADC24_Loop(void)
{
    static float32_t temp;
    if(nadc24IntFlag)
    {
     /* Read Current value */
     u32AdcData = SPI_ReadADCData();

    if (u32AdcData & 0x800000)
     u32AdcData += 0xFF000000;//Convert to a signed number.

    s32AdcData = (int32_t)u32AdcData;

    switch(nadc24_mode)
    {
    case NADC24_GET_CHTP://Internal temperature sensor.

    NADC24_RawData_Database[Average_Count] = s32AdcData;

    Average_Count++;
    /* Average value */
    if (Average_Count >= 16)
    {
         if(RegInput[FREE_RUN_REG] > 1)
         SPI_Send_ADC_Commnad(ADC_STOP_CONVERSION_CMD);

         arm_mean_q31(&NADC24_RawData_Database[0], 16, &NADC24_RawData);
         Calc_NADC24TP(NADC24_RawData);

         /* Finish */
         Average_Count = 0;
         if(RegInput[FREE_RUN_REG] > 1)
         {
         NADC24_Mode_Set(NADC24_GET_CH67);//Channel switching.
         }
    }
    break;

    case NADC24_GET_CH67://Thermocouple temperature sensor.

    NADC24_RawData_Database[Average_Count] = s32AdcData;

    Average_Count++;
    /* Average value */
     if (Average_Count >= 16)
     {
         Average_Count = 0;
         arm_mean_q31(&NADC24_RawData_Database[0], 16, &NADC24_RawData);

         temp = (float32_t)NADC24_RawData;
         //PGA = 32; Cold junction compensation.
```

```
            temp = ((1200*temp)/0x7fffff)/32 + TempColdVol;

            /* Finish */
            RegInput[TCK_REG]=(int16_t)(TC_Table_Convert(TS_Table, TCK_Temp_Min, temp)*10);

            printf("TC = %d,%d\n\n", NADC24_RawData,RegInput[TCK_REG]);
        }
        default :break;
        }
        nadc24IntFlag = 0;
        RegHold[NADC24_CH_REG] = nadc24_mode ;
        }
}
```

2.4 热电偶温度计算

热电偶的温度计算在前面已有介绍，这里展示一下根据热电偶的电势计算温度。先根据查表法得出当前温度整数位的电势区段，然后再通过比例计算温度的小数位，将整数位和小数位相加则可以得出浮点数的温度数据。下面程序仅仅是根据电势换算温度的功能，在测量绝对温度的时候需要进行冷端温度的补偿，补偿的计算过程在 *NADC24_Loop()* 函数中。

```
/*******************************************************************************
 *  @brief      TC Table Convert
 *  @param[in]  tc_table : Table pointer.
 *  @param[in]  temp_min : The lowest measurable temperature.
 *  @param[in]  voltage_mv : Measure the voltage value of the thermocouple.
 *  @return     Temperature value.
 *  @details    Calculate the temperature value based on the voltage lookup table.
*******************************************************************************/
float TC_Table_Convert( const float * tc_table, int16_t temp_min, float voltage_mv )
{
    uint16_t index_temp;
    float   tc_temp;

    //Searches for data locations in a table through indexes.
    index_temp = Get_TableIndex(tc_table, TCK_Table_Total,voltage_mv );

    //temp_min + index_temp + ( voltage_mv - table_vlotage ) / (The voltage difference
   between two indexes)
    tc_temp    = temp_min + index_temp + ( voltage_mv - * ( tc_table + index_temp ) ) /
    ( * ( tc_table + index_temp + 1 ) - * ( tc_table + index_temp ) );

    return (float)tc_temp;
}
```

2.5 连接测试

例程测试使用 Mthing 读取 Modbus 协议的数据。Mthing 是一款功能强大的 Modbus 协议分析软件，个人版可免费使用。更多信息可访问其官网：

http://www.gulink.cn/

1. 按照箭头顺序添加通讯端口，参考图 2-1：

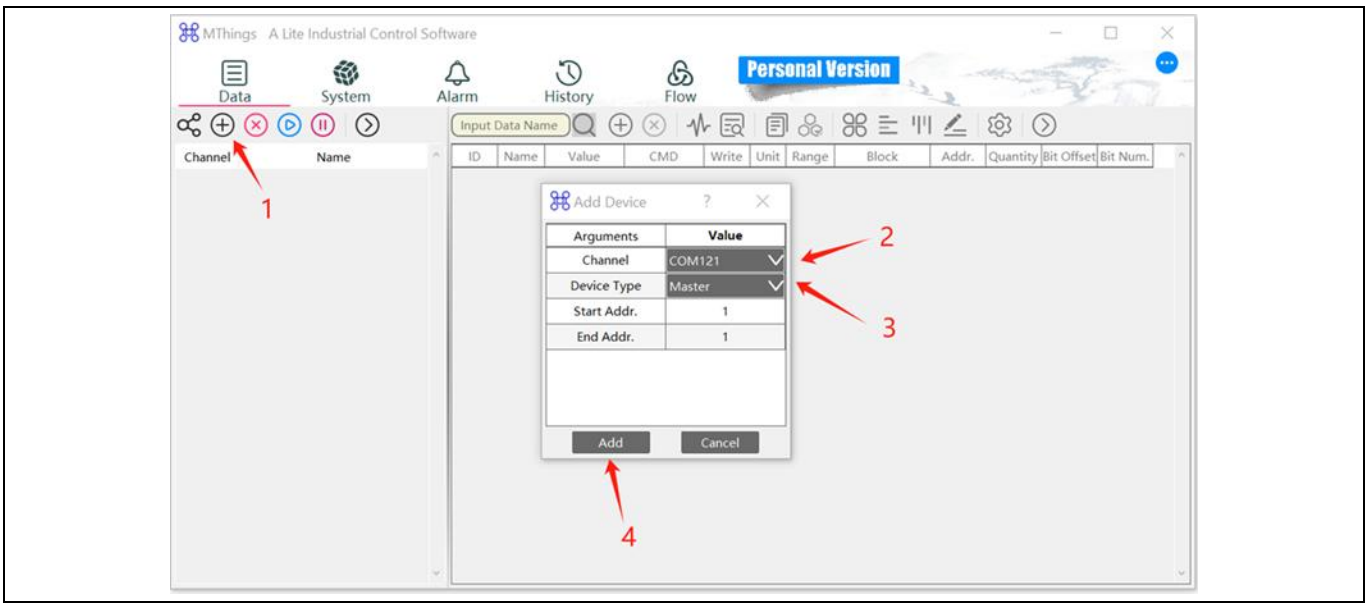

图 2-1 添加通讯端口

2. 按照箭头顺序设置通讯速率，参考图 2-2：

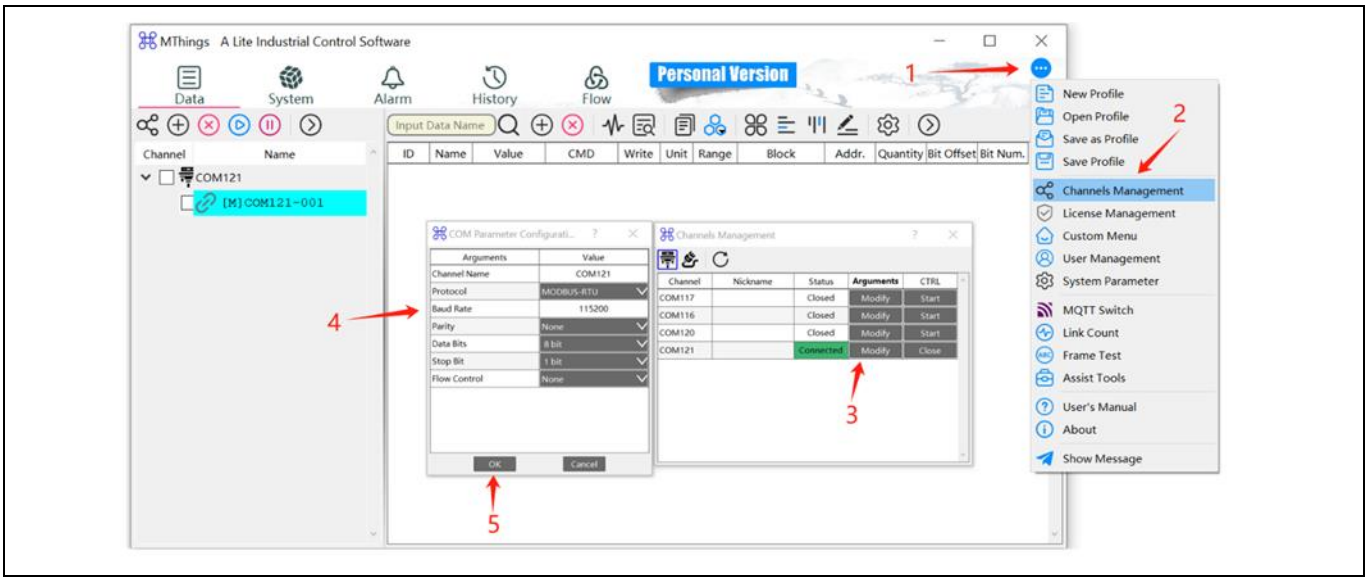

图 2-2 设置通讯速率

3. 添加读寄存器的参数，参考图 2-3：

   1) 添加读数据；

   2) 一次读 6 个寄存器；

   3) 寄存器类型为输入寄存器；

   4) 读寄存器起始地址为 1000；

   5) 添加结束。

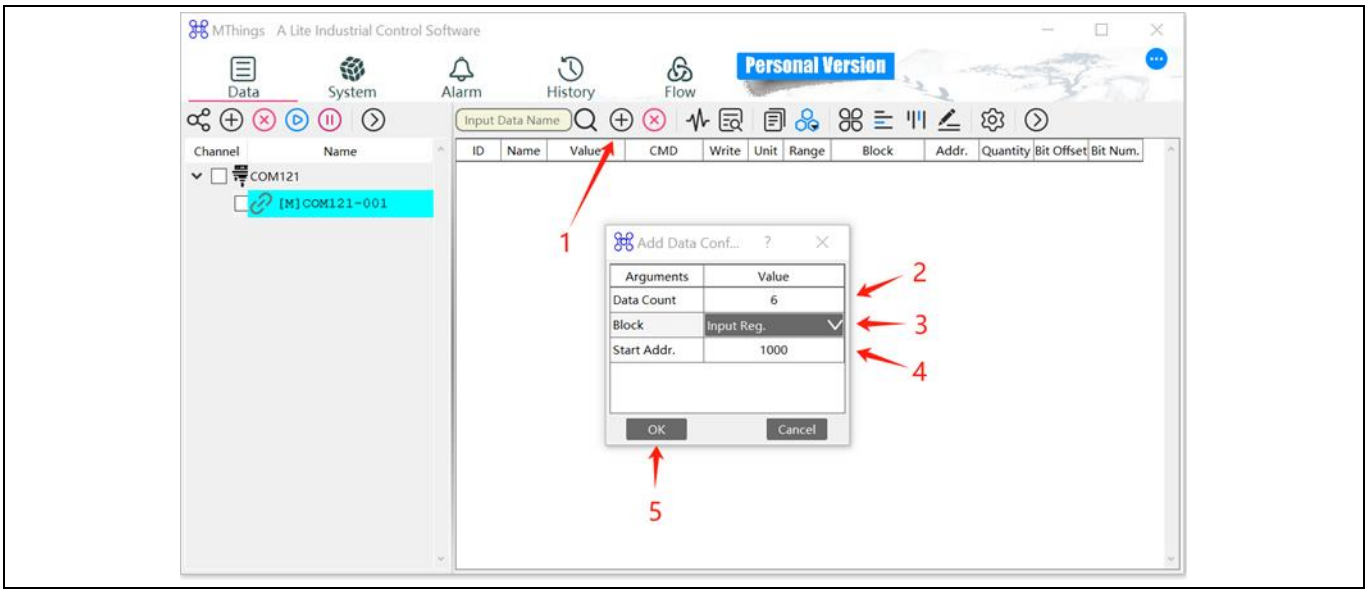

图 2-3 添加读寄存器

4. 开始数据轮询，参考图 2-4：

   - 当前热电偶测试温度十进制数据为 259，实际温度为 25.9 ℃。
   - NADC24 内部的温度十进制数据为 263，实际温度为 26.3 ℃。

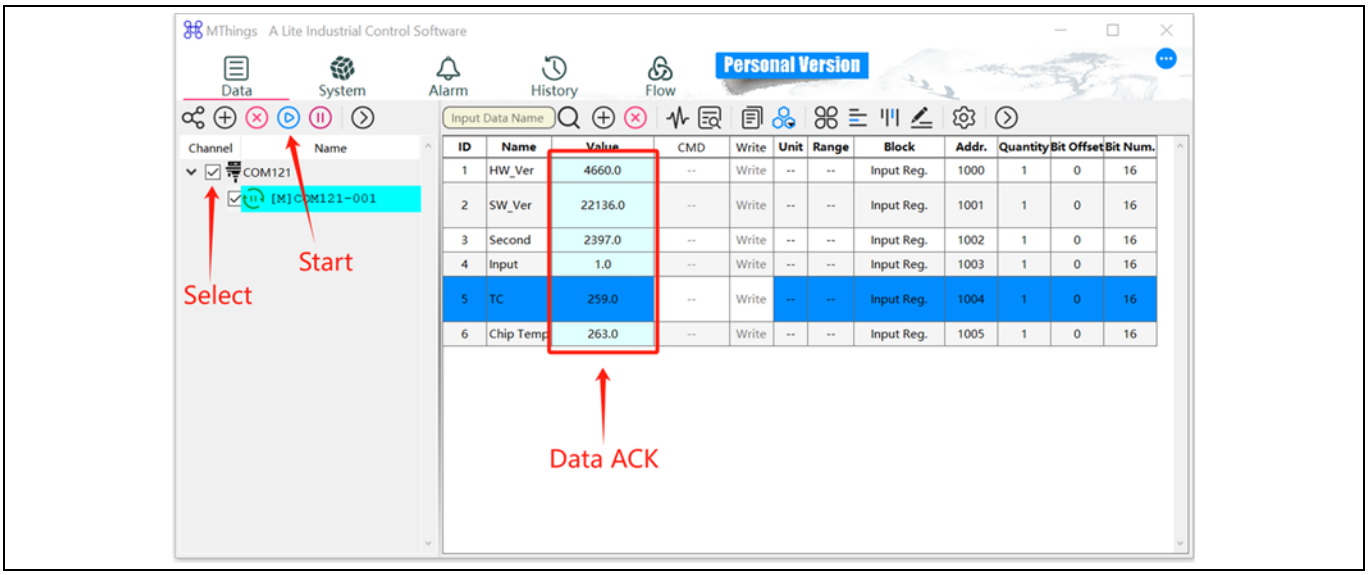
MThings A Lite Industrial Control Software
Data
System
Alarm
History
Flow
Personal Version
Input Data Name
Channel
Name
COM121
[M]COM121-001
Start
Select
ID Name Value CMD Write Unit Range Block Addr. Quantity Bit Offset Bit Num.
1 HW_Ver 4660.0 -- Write -- -- Input Reg. 1000 1 0 16
2 SW_Ver 22136.0 -- Write -- -- Input Reg. 1001 1 0 16
3 Second 2397.0 -- Write -- -- Input Reg. 1002 1 0 16
4 Input 1.0 -- Write -- -- Input Reg. 1003 1 0 16
5 TC 259.0 -- Write -- -- Input Reg. 1004 1 0 16
6 Chip Temp 263.0 -- Write -- -- Input Reg. 1005 1 0 16
Data ACK

图 2-4 开始读数据

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - CM2003_Series_BSP_CMSIS_V3.00.000
- IDE 版本
  - Keil uVision 5.37@ Arm Compiler 6
- PC APP 版本
  - MThings V0.8.0.6

3.2 硬件需求

- 电路组件
  - CM2003_ModbusRTU_NADC24_V0.1
  - USB 转 RS485 适配器
- 线路示意图
  - USB 转 RS485 设备接入电脑，以实现上位机的软件的数据读写。
  - 将 ModbusRTU 从机 RS485 连接到 USB 转出的 RS485，注意 RS485 信号线的 A/B 极性。

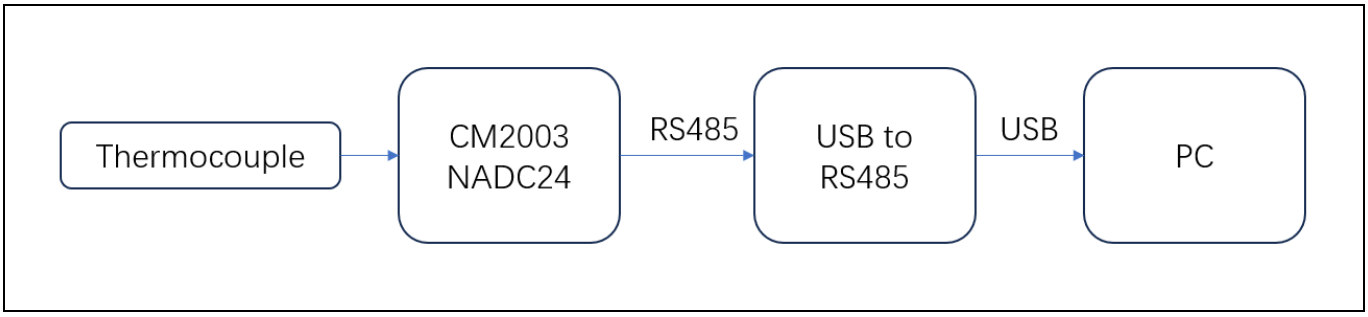

图 3-1 线路示意图

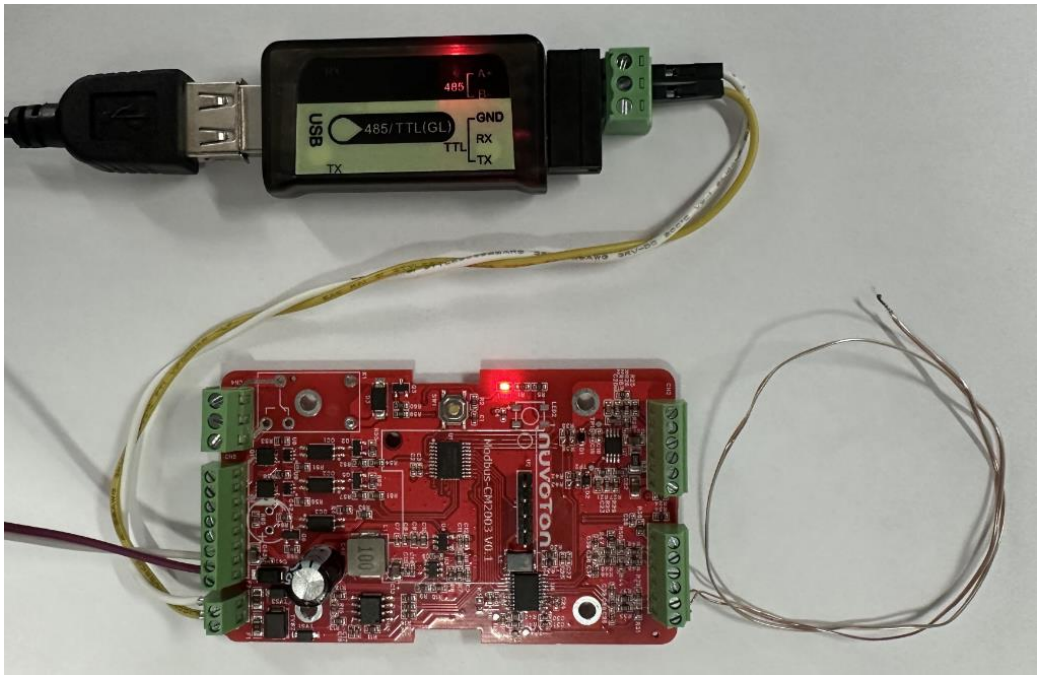

图 3-2 实物接线

4. 目录信息

目录结构如下图所示：

EC_CM2003_ModbusRTU_Thermocouple_V1.0	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *Modbus_Thermocouple.uvprojx*
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

6. 修订纪录

Date	Revision	Description
2025.05.28	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.