# Thermocouple Temperature Collector Based on ModbusRTU Protocol

Example Code Introduction for 32-bit NuMicro® Family

## Document Information

| | |
|---|---|
| Application | This example code introduces how to use the USCI interface of the CM2003 series microcontroller (MCU) to connect the 24-bit Analog-to-Digital Converter NADC24 to collect the temperature signal of the thermocouple sensor. |
| BSP Version | CM2003_Series_BSP_CMSIS_V3.00.000 |
| Hardware | 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. Overview

This example code introduces how the CM2003 series microcontroller (MCU) reads the differential channel of the 24-bit Analog-to-Digital Converter NADC24 through the USCI interface to collect the temperature signal of the external thermocouple sensor. The temperature signal is read and transmitted to the USB to RS485 adapter using the Modbus protocol over the RS485 interface. The USB to RS485 adapter is then transferred to the PC for analysis.

The Modbus protocol uses the open source FreeModbus protocol stack. FreeModbus is widely used in the development of embedded Modbus devices due to its open source, lightweight and easy portability. Developers can customize and optimize it according to project requirements.

For more information about FreeModbus, please refer to its official website:

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

## 1.1 CM2003 Introduction

This example requires three basic functions of CM2003: UART, SPI, TIMER.

- UART: The interface is responsible for sending and receiving data of Modbus protocol through RS485.
- SPI: The interface is responsible for reading and writing data from NADC24. Since CM2003 does not have a standard SPI interface, the SPI function of the USCI interface is used to implement it. NADC24 has 8-bit and 24-bit data modes, but USCI can only support 8-bit and 16-bit modes. Therefore, the USCI program uniformly reads and writes data in 8-bit mode.
- TIMER: TIMER0 is used to determine communication timeout. The Modbus protocol requires that each frame of data interval must have a time of ≥ 3.5-bit signals. TIMER1 is used to generate second counts.

## 1.2 Modbus Introduction

Modbus uses a master-slave architecture and has three protocol types:

- Modbus RTU (based on serial communication, such as RS-485/RS-232, binary encoding, efficient and compact).
- Modbus ASCII (based on serial communication, ASCII character encoding, which is highly readable but inefficient).
- Modbus TCP/IP (based on Ethernet, using TCP ports, suitable for modern industrial networks).

The main contents of the protocol are:

Device address + Command + Register address + Data + CRC check

There are eight commands listed in Table 1-1:

| Command(decimal) | Operational | Data Types |
|---|---|---|
| 1 | Read Coils | 1-bit (Switching quantity) |
| 2 | Read Discrete Inputs | 1-bit (Read only) |
| 3 | Read Holding Registers | 16-bit (Read and Write) |
| 4 | Read Input Registers | 16-bit (Read only) |
| 5 | Write Single Coil | 1-bit (Switching quantity) |
| 6 | Write Single Register | 16-bit (Read and Write) |
| 15 | Write Multiple Coils | 1-bit (Bulk) |
| 16 | Write Multiple Registers | 16-bit (Bulk) |

Table 1-1 Modbus Command List

This example uses four commands in the Modbus protocol: 3, 4, 6, 16. The Modbus communication format defined in this example is present in Table 1-2:

| Communication Parameters | Parameter Value |
|---|---|
| Address | 1 |
| Baud_rate | 115200 |
| Data_bits | 8 |
| Stop_bits | 1 |
| Parity | None |
| Flow_control | None |

Table 1-2 Communication Format

Register definition in Table 1-3:

| Register Address (decimal) | Register Type | Description | Data Format |
|---|---|---|---|
| 1000 | Read Input Registers (Read only) | Hardware Version | 16-bit HEX |
| 1001 | | Software Version | 16-bit HEX |
| 1002 | | Seconds count | 1 Second |
| 1003 | | IO input | 0/1 |
| 1004 | | Thermocouple temperature | 0.1°C |
| 1005 | | NADC24 temperature | 0.1°C |
| 2000 | Holding Registers (Read and Write) | LED Status | 0/1 |
| 2001 | | IO output | 0/1 |
| 2002 | | NADC24 channel | same as NADC24 channel register |
| 2003 | | MCU work mode | reserve |

Table 1-3 Register Data

For more information about Modbus, please refer to the protocol document of the Modbus organization:

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

## 1.3 Thermocouple Temperature Acquisition Principle

When two different metal conductors (or semiconductors) are connected at both ends to form a closed loop, if there is a temperature difference (ΔT) between the two connection points (called the hot end and the cold end), an electromotive force (voltage) will be generated in the loop. This phenomenon is called the Seebeck Effect. The voltage generated is called thermoelectric potential, and its magnitude is proportional to the temperature difference, and the relationship is:

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

Explanation:

- V: Thermoelectric potential (Unit: mV).
- α: Seebeck coefficient (Material properties, Unit: μV/°C).
- ΔT: The temperature difference between the hot end and the cold end (Unit: °C).

Basic Methods of Thermocouple Measurements:

- Hot end (measuring end): exposed to the temperature environment to be measured, called $T_1$.
- Cold end (reference end): Maintain a known constant temperature (such as 0°C) or correct it through a compensation circuit, called $T_2$.
- Wire material: two different metal conductors.

Thermocouples measure the thermoelectric potential V to calculate the hot end temperature $T_1$, and the cold end temperature $T_2$ must be known.

If the cold end temperature of the thermocouple is not 0°C but $T_2$, the relationship between the thermoelectric potential V ($T_1$, $T_2$) and the standard scale (cold end is 0°C) is:

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

### 1.3.1 Thermocouple Classification

Thermocouples are available in a variety of specifications depending on the conductor material.

- K-type thermocouple: Made of nickel-chromium-nickel-silicon material.

  It is suitable for the temperature range of -200°C to 1300°C and is widely used in industrial fields.

- J-type thermocouple: Made of iron-constantan material.

Applicable to the temperature range of -200°C to 750°C and suitable for use in oxidizing gas environments.

- E-type thermocouple: Made of nickel-chromium-copper-nickel material.

  It is suitable for the temperature range of -200°C to 900°C and has a high thermoelectric potential.

- T-type thermocouple: Made of copper-constantan material.

  Applicable to the temperature range of -200°C to 350°C, suitable for low temperature measurement.

- N-type thermocouple: Made of nickel-chromium-silicon-nickel-silicon-magnesium material.

  It is suitable for the temperature range of -200°C to 1300°C and has good oxidation resistance.

This example uses the widely used K-type thermocouple and complies with the ITS-90 international temperature scale. For more information about ITS-90, please refer to the official definition:

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

Correspondence between K-type temperature and electric potential:

| | when 0°C corresponds to 0 mV | | | |
|---|---|---|---|---|
| -270°C | <=== | 0°C | ===> | 1372°C |
| -6.458 mV | <=== | 0 mV | ===> | 54.886 mV |

For detailed temperature-voltage correspondence, please refer to the link file:

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

### 1.3.2 Thermocouple Temperature Calculation Method

There are two methods for calculating the temperature of thermocouples, the table lookup method and the calculation method. Both calculation methods are recognized by ITS-90.

- Table lookup method: Usually when the processor capability is weak, this example uses the table lookup method to calculate the temperature.
- Calculation method: The temperature can be calculated through a polynomial equation.

### 1.3.3 Steps to measure thermoelectric temperature:

1. Read the NADC24 internal temperature calibration value.
2. Calculate the NADC24 internal temperature sensor temperature (cold end temperature) based on the calibration value.
3. Convert the cold end temperature into the cold end voltage.

4. Measure the thermocouple hot end voltage.

5. Calculate the absolute temperature through the hot end voltage and the cold end voltage.

## 1.4 NADC24 Introduction

The potential change of thermocouple is very small, and conventional ADC cannot accurately collect it. High resolution and differential channels are required to accurately collect the potential change of thermocouple. The NADC24 used in this example has 3 groups of differential channels, and the internal temperature sensor can be used for cold end temperature collection; CH6 and CH7 form a group of differential channels to collect hot end temperature.

The NADC24 generates an interrupt signal every time it samples, which can be set to continuous sampling or single sampling. In this example, a gain of 32 times and a sampling rate of 32 kbps are used for continuous sampling.

# 2. Code Description

The code is mainly divided into four parts:

- Main framework of the program: responsible for initialization and data processing.
- FreeModbus protocol stack: responsible for Modbus protocol communication parameter setting and data transmission and reception.
- NADC24 voltage measurement: including IC initialization, calibration program, and data acquisition.
- Thermocouple temperature calculation: calculation of thermocouple temperature.

## 2.1 Main Program Structure

The main program starts to initialize the system, including the initialization of the internal interface of CM2003, the initialization of the Modbus protocol in the *SYS_Init()* function, the initialization of NADC24, and the initialization of the Modbus protocol register. In the main loop, *eMBPoll()* is responsible for parsing the Modbus protocol, and *NADC24_Loop()* is triggered by the interrupt flag when the conversion is detected.

```
/*---------------------------------------------------------------*/
/* 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 Protocol Stack

After the FreeModbus protocol stack is transplanted, you only need to pay attention to the processing of the four types of register data in the *user_modbus.c* file. Only two registers, input register and holding register, are used in this example. The input register only sends data to the outside, and the holding register can be read and written. If the user needs non-volatility, add the Flash write function to the holding register.

```
/*********************************************************************************
 *  @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: Register address.
 *                    usNRegs: Number of register operations.
 *                   eMode: 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 Voltage Measurement

The NADC24 has already done the calibration of the measurement circuit during initialization. The internal temperature sensor needs to be sampled for the first time when the system starts (cold end sampling). After the sampling is completed, the channel is switched to the thermocouple sampling channel for continuous sampling (hot end sampling). Both types of sampling are averaged 16 times, and the sampling rate is 32 kbps. After 16 samples are completed, the temperature is calculated and placed in the corresponding input register of Modbus. It should be noted that after 16 averages, the actual temperature update rate is about 0.5 seconds.

```
/*****************************************************************************
 *  @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 Thermocouple Temperature Calculation

The temperature calculation of thermocouples has been introduced before. Here will show how to calculate the temperature based on the potential of thermocouples. First, the potential segment of the integer digit of the current temperature is obtained by the table lookup method, and then the decimal digit of the temperature is calculated by proportion. The floating-point temperature data can be obtained by adding the integer digit and the decimal digit. The following program is only based on the function of converting potential to temperature. When measuring absolute temperature, cold end temperature compensation is required. The compensation calculation process is in the *NADC24_Loop()* function.

```
/*******************************************************************************
 *  @brief       TC Table Convert
 *  @param[in]   tc_table      : Table pointer.
 *               temp_min      : The lowest measurable temperature.
 *               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 Connection Test

This example code test uses Mthing to read Modbus protocol data. Mthing is a powerful Modbus protocol analysis software, and the personal version is free to use. For more information, visit its official website:

http://www.gulink.cn/.

1. Add communication ports in the order of the arrows as Figure 2-1.

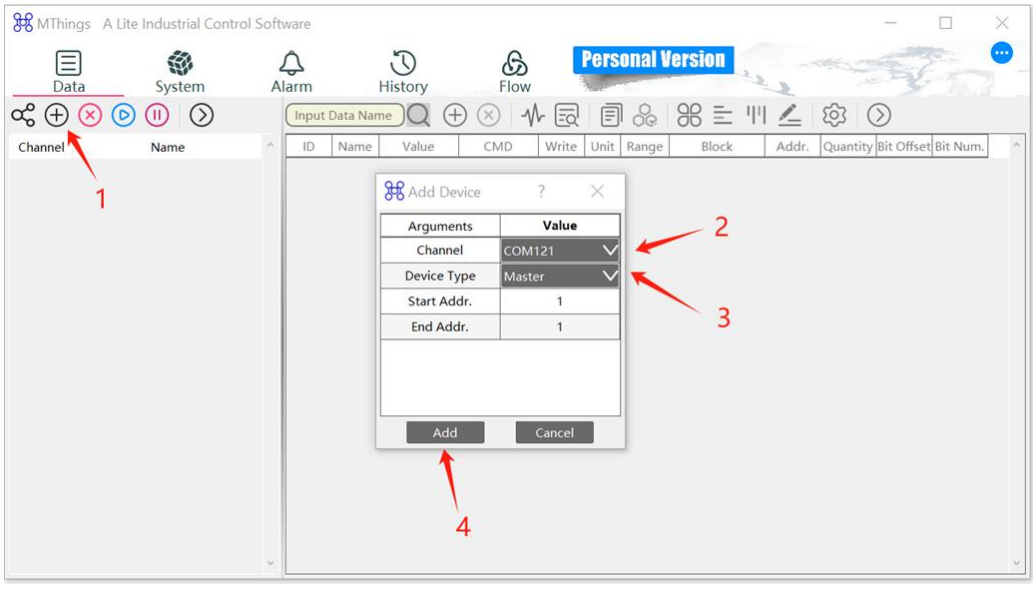


Figure 2-1 Adding a Communication Port

2. Set the communication rate according to the arrow sequence as shown in Figure 2-2.

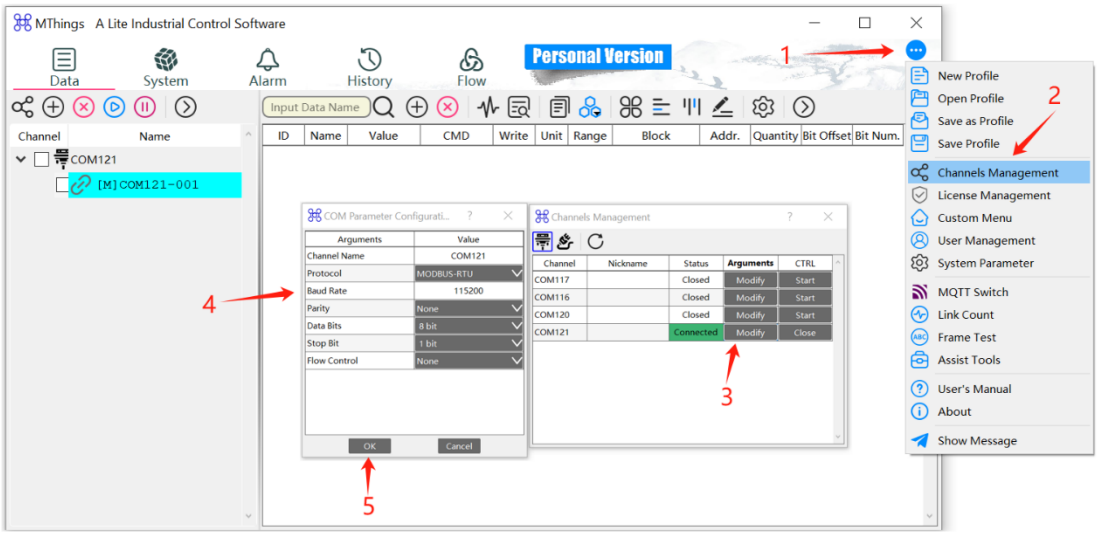


Figure 2-2 Set the Communication Rate

3. Added parameter for reading registers as Figure 2-3:

   1) Add register data.
   2) Read 6 registers at a time.
   3) Register type is input register.
   4) The starting address for reading register is 1000.
   5) Add the end.

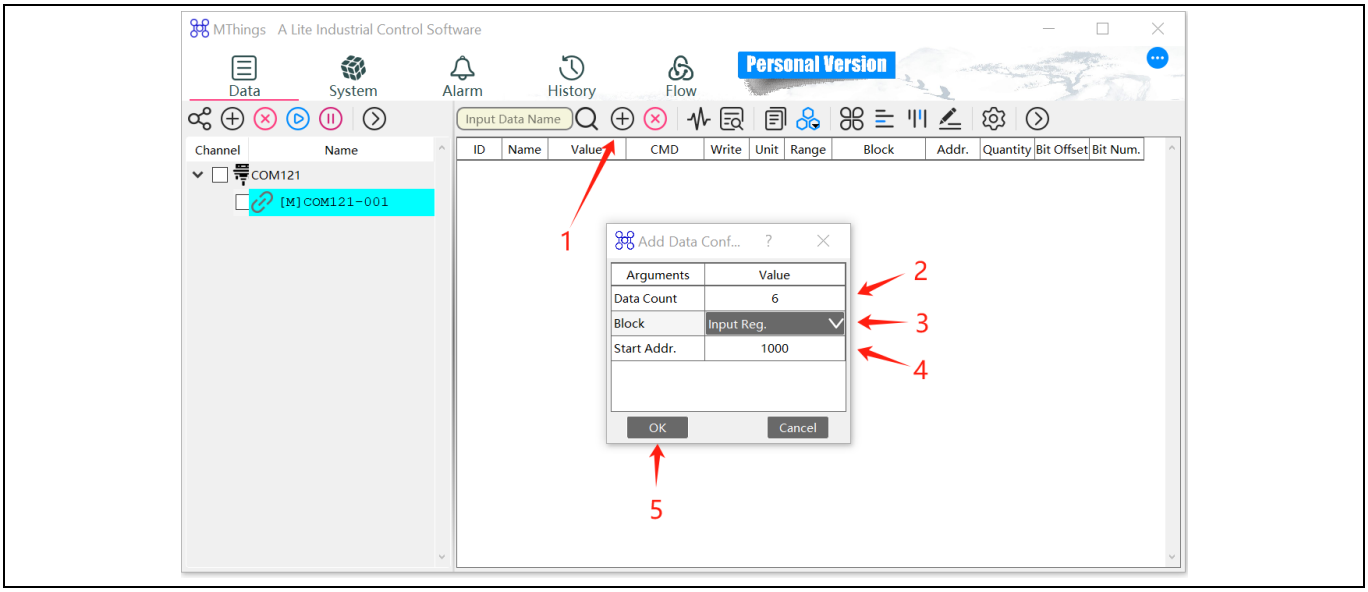


Figure 2-3 Add Parameter for Input Register

4. Start data polling as Figure 2-4:

   - The current thermocouple test temperature decimal data is 259, and the actual temperature is 25.9°C.
   - The decimal temperature data inside NADC24 is 263, and the actual temperature is 26.3°C.

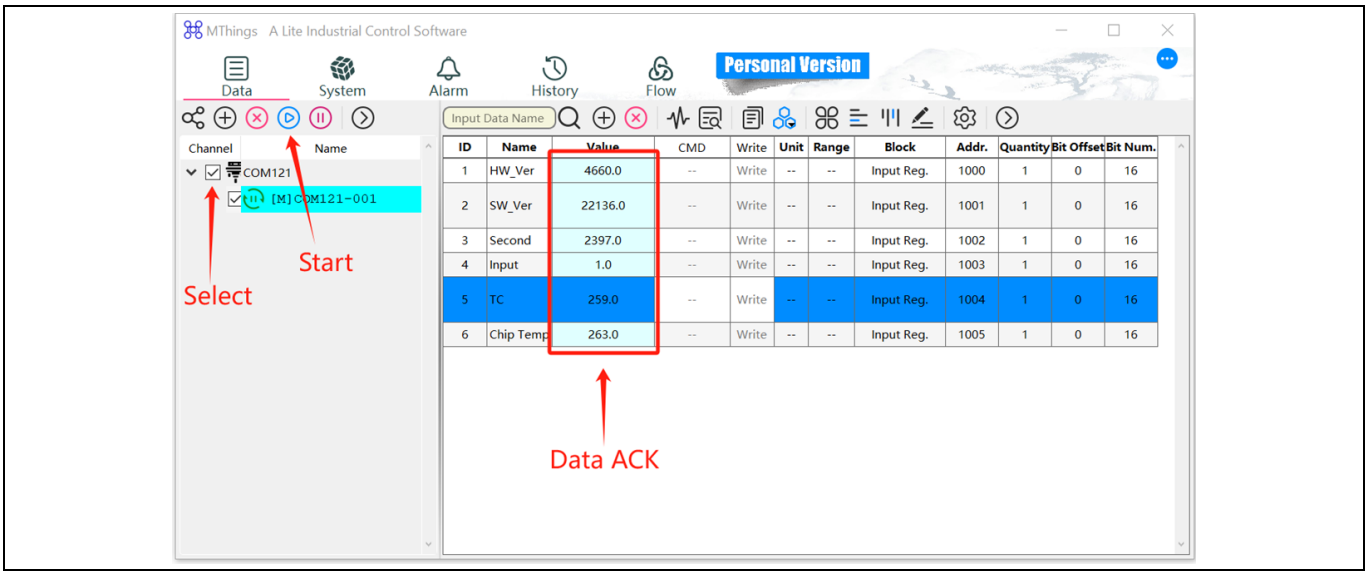


Figure 2-4 Start Reading

# 3. Software and Hardware Requirements

## 3.1 Software Requirements

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

## 3.2 Hardware Requirements

- Circuit Components
  - CM2003_ModbusRTU_NADC24_V0.1
  - USB to RS485 adapter
- Pin Connect
  - The USB to RS485 adapter is connected to the computer to enable the data reading and writing of the host computer software.
  - Connect the ModbusRTU slave RS485 to the USB to RS485 adapter. Pay attention to the A/B polarity of the RS485 signal line.

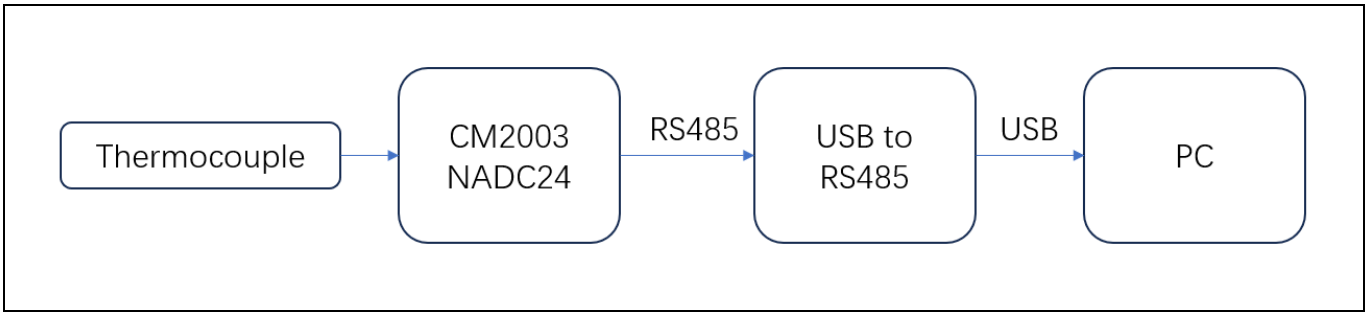


Figure 3-1 Pin Connect

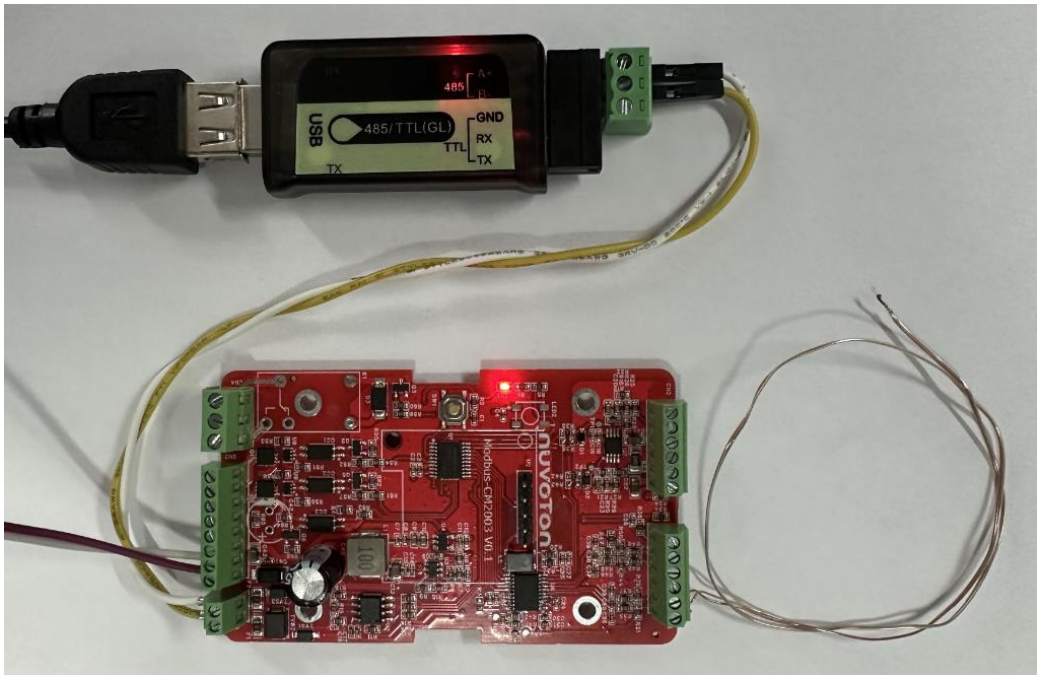


Figure 3-2 Physical Wiring

# 4. Directory Information

The directory structure is shown below.

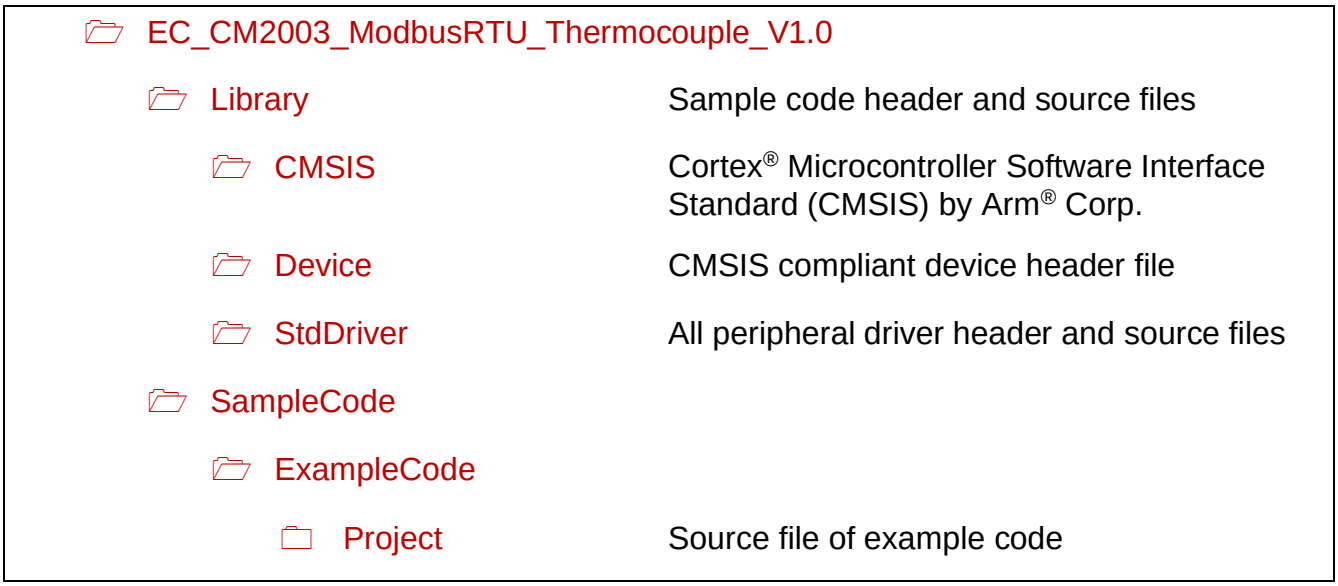


Figure 4-1 Directory Structure

# 5. Excample Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *Modbus_Thermocouple.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

# 6. Revision History

| Date | Revision | Description |
|---|---|---|
| 2025.05.28 | 1.00 | Initial version. |

## 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.*