使用LwIP实现Modbus TCP功能

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 LwIP 资料库来实作 Modbus TCP 的工业控制网络应用，用户可以藉由 Modbus TCP 协议，透过网络服务器控制 M467 的状态。
BSP 版本	M460 Series BSP CMSIS V3.00.001
开发平台	NuMaker-M467HJ V1.0

1. 概述

本范例代码使用 LwIP 资料库来实作 Modbus TCP 的工业控制网络应用，用户可以藉由 Modbus TCP 协议，透过网络服务器控制 M467 的状态。

Modbus 是一种工业通讯协议，用于自动化系统中，具有监控系统和远程数据收集等功能，Modbus 通讯协议常用的通讯接口包括 RS-232、RS-485 和以太网等，它定义了一组命令和数据格式，使主机能够向从机发送指令，这些指令可以用于读取和写入从机数据。

Modbus 主要用于监控和控制各种设备，如 PLC（可编程逻辑控制器）、传感器、阀门、伺服驱动器和人机接口等。它提供了一种标准化的通讯方式，使不同供货商的设备能够互相通讯，提高自动化系统的效率和可靠性。

1.1 原理

Modbus 协议定义了多种指令码，用于控制从机的行为，以下是 Modbus 协议中常见的指令码：

指令码	指令名称
01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Register
05	Write Single Coil
06	Write Single Register
15	Write Multiple Coils
16	Write Multiple Registers

表 1-1 指令码

以下针对这些控制指令，说明主机如何控制从机：

- **Read Coils（读取线圈）**：这个指令用于读取单个位数据（通常表示状态）。
- **Read Discrete Inputs（读取离散输入）**：这个指令类似于 Read Coils，主要用于读取离散输入的状态，通常表示开关或按钮的状态。
- **Read Holding Registers（读取保持缓存器）**：这个指令用于读取 16 位的数值数据。
- **Read Input Register（读取输入缓存器）**：这个指令类似于 Read Holding Registers，但是用于读取输入寄存器的值。输入寄存器通常用于储存传感器的数据。

- **Write Single Coil（写入单个线圈）**：这个指令用于写入单个位数据。
- **Write Single Register（写入单个缓存器）**：这个指令用于写入单个 16 位的数值数据。
- **Write Multiple Coils（写入多个线圈）**：这个指令用于写入多个位数据到不同线圈中。
- **Write Multiple Registers（写入多个缓存器）**：这个指令用于写入多个 16 位的数值数据到不同缓存器中。

这些控制指令提供了主机与从机之间的基本控制功能，可以用于监控和控制自动化系统中的各种设备，主机可以根据需要使用这些指令进行读取或写入操作，从而实现对从机设备的控制和调节。

1.2 执行结果

本范例代码执行编译和刻录后，使用 Modbus TCP 的 PC 工具 Modbus Poll 程序进行通讯

Modbus Poll 操作如下：

1. 点选 **Connect** 图标，设定 IP 地址为 192.168.1.2，并且与 PC 位于相同网域。

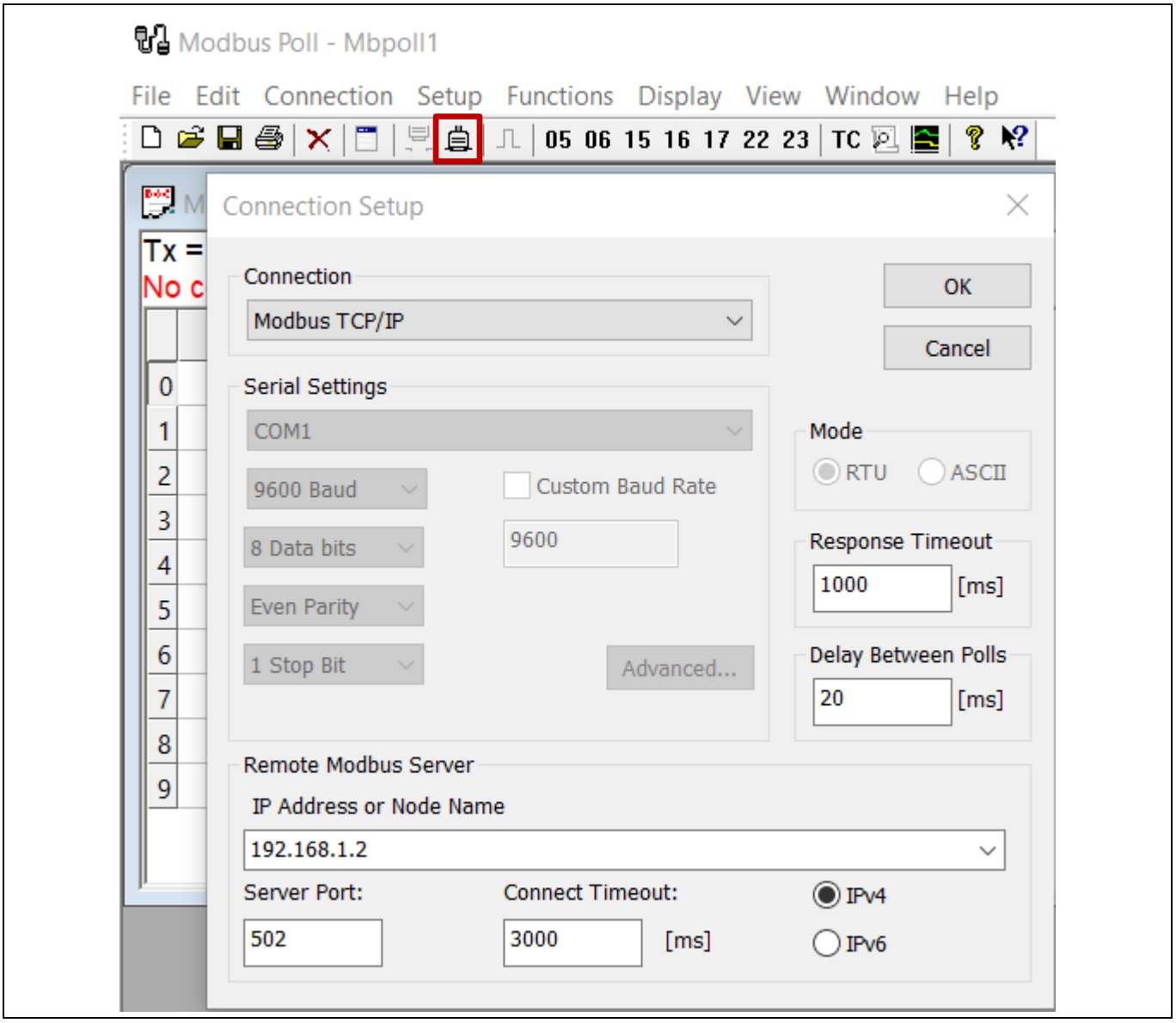

图 1-1 连接设定

2. 双击字段 0 的数值，可开启指令窗口，设定 Value 为 151 并按下 **Send**，即可发送数据。

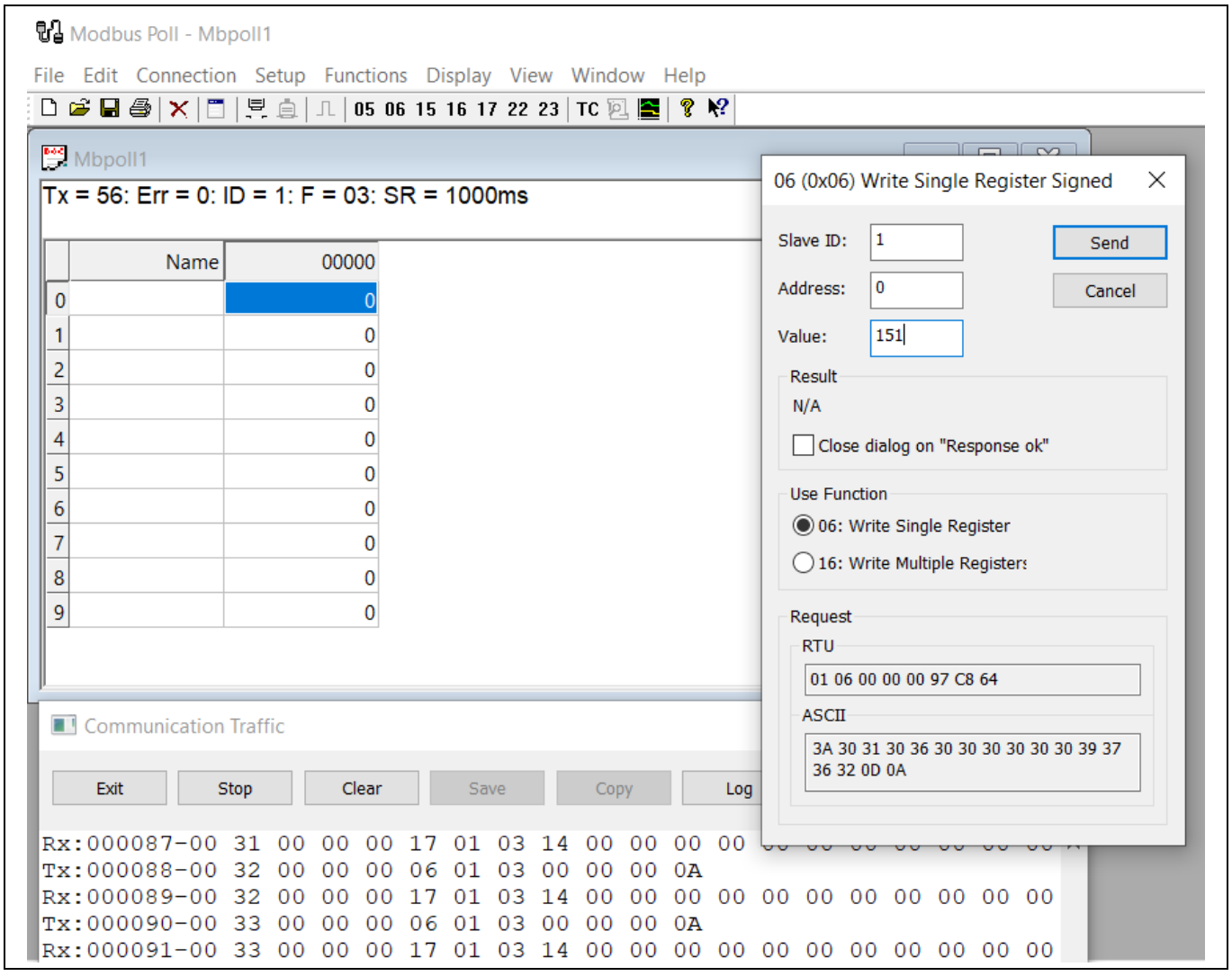

图 1-2 发送数据

通讯结果如图 1-3 所示。

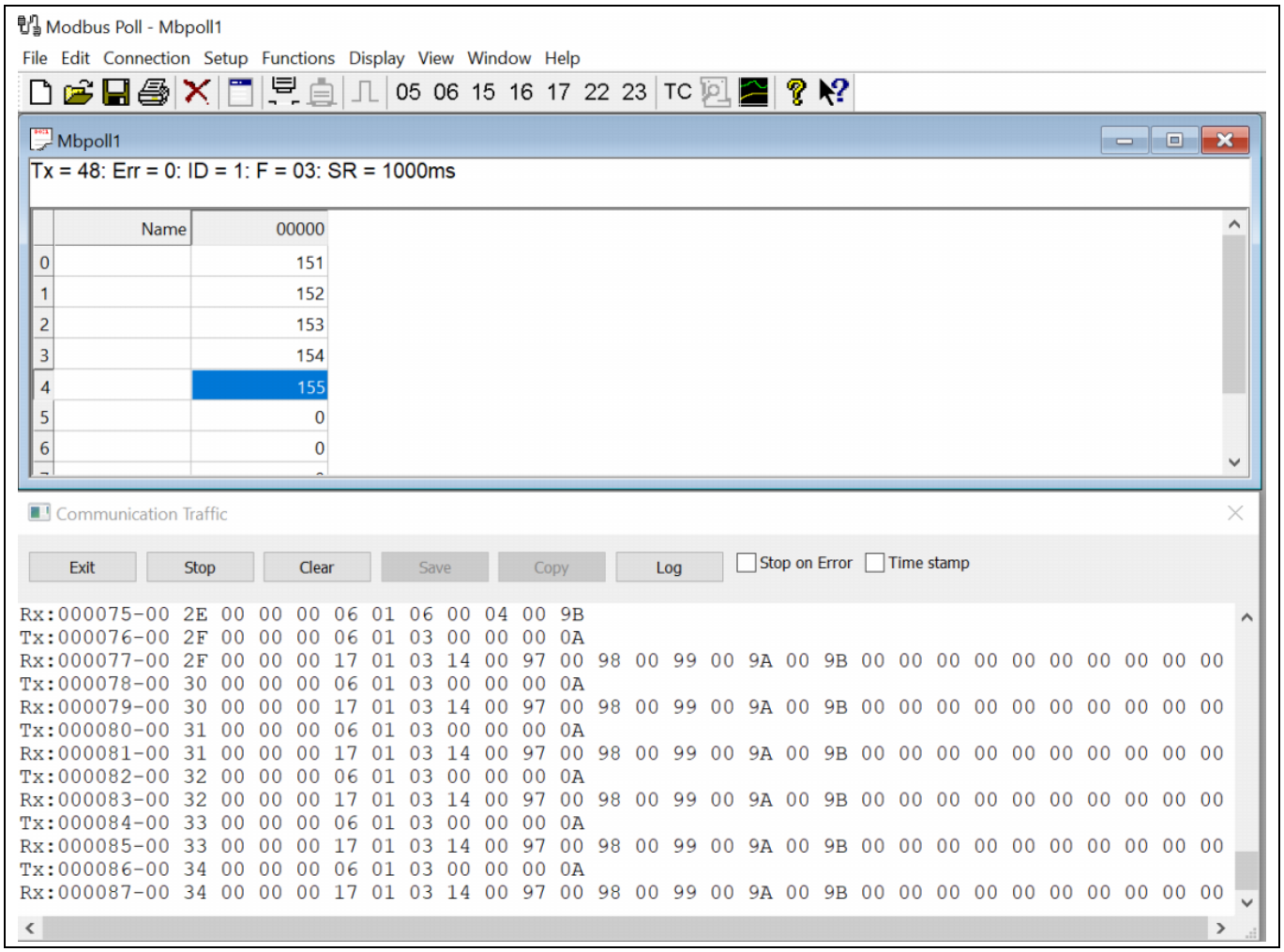

图 1-3 Modbus TCP 通讯数据

2. 代码介绍

设定 MCU 硬件状态，并建立 FreeRTOS 上的 TCP Task，执行网络应用，代码位于 *main.c*。

```
int main(void)
{
    /* Configure the hardware ready to run the test. */
    prvSetupHardware();

    xTaskCreate( vTcpTask, "TcpTask", TCPIP_THREAD_STACKSIZE, NULL,
mainCHECK_TASK_PRIORITY, NULL );

    printf("FreeRTOS is starting ...\n");

    /* Start the scheduler. */
    vTaskStartScheduler();

    /* If all is well, the scheduler will now be running, and the following line
    will never be reached.  If the following line does execute, then there was
    insufficient FreeRTOS heap memory available for the idle and/or timer tasks
    to be created.  See the memory management section on the FreeRTOS web site
    for more details. */
    for( ;; );
}
```

初始化系统后，执行 TCP Task，设定网络 IP 等相关参数，并启动 Modbus Task。

```
static void vTcpTask( void *pvParameters )
{
    ip_addr_t ipaddr;
    ip_addr_t netmask;
    ip_addr_t gw;

    IP4_ADDR(&gw, 192, 168, 1, 1);          //Set gateway IP 192.168.1.1
    IP4_ADDR(&ipaddr, 192, 168, 1, 2);      //Set IP address 192.168.1.2
    IP4_ADDR(&netmask, 255, 255, 255, 0);   //Set net mask 255.255.255.0

    tcpip_init(NULL, NULL);

    netif_add(&netif, &ipaddr, &netmask, &gw, NULL, ethernetif_init, tcpip_input);

    netif_set_default(&netif);
    netif_set_up(&netif);

    printf("[ TCP_Modbus ] \n");
    printf("IP address:      %s\n", ip4addr_ntoa(&netif.ip_addr));
    printf("Subnet mask:     %s\n", ip4addr_ntoa(&netif.netmask));
    printf("Default gateway: %s\n", ip4addr_ntoa(&netif.gw));

    tcp_netconn_init();

    vTaskSuspend( NULL );
```

```
}

static void tcp_netconn_thread(void *arg)
{
    eMBErrorCode    xStatus;

    for( ;; )
    {
        //Initialize Modbus stack
        if( eMBTCPInit( MB_TCP_PORT_USE_DEFAULT ) != MB_ENOERR )
        {
            printf("can't initialize modbus stack!\r\n");
        }
        else if( eMBEnable(  ) != MB_ENOERR )   //Enable Modbus stack
        {
            printf("can't enable modbus stack!\r\n");
        }
        else
        {
            do
            {
                xStatus = eMBPoll(  );      //Polling Modbus event
            }
            while( xStatus == MB_ENOERR );
        }
        /* An error occured. Maybe we can restart. */
        ( void )eMBDisable(  );
        ( void )eMBClose(  );
    }
}
```

底下接着介绍 FreeModbus 资料库提供的指令处理代码：

- Coil 线圈的 Callback Function，处理线圈相关的指令。

```
eMBErrorCode eMBRegCoilsCB(UCHAR * pucRegBuffer, USHORT usAddress,
        USHORT usNCoils, eMBRegisterMode eMode)
{
    eMBErrorCode    eStatus = MB_ENOERR;
    USHORT          iRegIndex , iRegBitIndex , iNReg;
    UCHAR *         pucCoilBuf;
    USHORT          COIL_START;
    USHORT          COIL_NCOILS;
    USHORT          usCoilStart;
    iNReg =  usNCoils / 8 + 1;

    pucCoilBuf = ucSCoilBuf;
    COIL_START = S_COIL_START;
    COIL_NCOILS = S_COIL_NCOILS;
    usCoilStart = usSCoilStart;

    /* it already plus one in modbus function method. */
    usAddress--;
```

```
    if( ( usAddress >= COIL_START ) &&
        ( usAddress + usNCoils <= COIL_START + COIL_NCOILS ) )
    {
        iRegIndex = (USHORT) (usAddress - usCoilStart) / 8;
        iRegBitIndex = (USHORT) (usAddress - usCoilStart) % 8;
        switch ( eMode )
        {
        /* read current coil values from the protocol stack. */
        case MB_REG_READ:
            while (iNReg > 0)
            {
                *pucRegBuffer++ = xMBUtilGetBits(&pucCoilBuf[iRegIndex++],
                        iRegBitIndex, 8);
                iNReg--;
            }
            pucRegBuffer--;
            /* last coils */
            usNCoils = usNCoils % 8;
            /* filling zero to high bit */
            *pucRegBuffer = *pucRegBuffer << (8 - usNCoils);
            *pucRegBuffer = *pucRegBuffer >> (8 - usNCoils);
            break;

            /* write current coil values with new values from the protocol stack. */
        case MB_REG_WRITE:
            while (iNReg > 1)
            {
                xMBUtilSetBits(&pucCoilBuf[iRegIndex++], iRegBitIndex, 8,
                        *pucRegBuffer++);
                iNReg--;
            }
            /* last coils */
            usNCoils = usNCoils % 8;
            /* xMBUtilSetBits has bug when ucNBits is zero */
            if (usNCoils != 0)
            {
                xMBUtilSetBits(&pucCoilBuf[iRegIndex++], iRegBitIndex, usNCoils,
                        *pucRegBuffer++);
            }
            break;
        }
    }
    else
    {
        eStatus = MB_ENOREG;
    }
    return eStatus;
}
```

- Discrete Inputs 离散输入的 Callback Function，处理离散输入相关的指令。

```
eMBErrorCode eMBRegDiscreteCB( UCHAR * pucRegBuffer, USHORT usAddress, USHORT
usNDiscrete )
{
    eMBErrorCode    eStatus = MB_ENOERR;
```

```
    USHORT          iRegIndex , iRegBitIndex , iNReg;
    UCHAR *         pucDiscreteInputBuf;
    USHORT          DISCRETE_INPUT_START;
    USHORT          DISCRETE_INPUT_NDISCRETES;
    USHORT          usDiscreteInputStart;
    iNReg =  usNDiscrete / 8 + 1;

    pucDiscreteInputBuf = ucSDiscInBuf;
    DISCRETE_INPUT_START = S_DISCRETE_INPUT_START;
    DISCRETE_INPUT_NDISCRETES = S_DISCRETE_INPUT_NDISCRETES;
    usDiscreteInputStart = usSDiscInStart;

    /* it already plus one in modbus function method. */
    usAddress--;

    if ((usAddress >= DISCRETE_INPUT_START)
            && (usAddress + usNDiscrete    <= DISCRETE_INPUT_START +
DISCRETE_INPUT_NDISCRETES))
    {
        iRegIndex = (USHORT) (usAddress - usDiscreteInputStart) / 8;
        iRegBitIndex = (USHORT) (usAddress - usDiscreteInputStart) % 8;

        while (iNReg > 0)
        {
            *pucRegBuffer++ = xMBUtilGetBits(&pucDiscreteInputBuf[iRegIndex++],
                    iRegBitIndex, 8);
            iNReg--;
        }
        pucRegBuffer--;
        /* last discrete */
        usNDiscrete = usNDiscrete % 8;
        /* filling zero to high bit */
        *pucRegBuffer = *pucRegBuffer << (8 - usNDiscrete);
        *pucRegBuffer = *pucRegBuffer >> (8 - usNDiscrete);
    }
    else
    {
        eStatus = MB_ENOREG;
    }

    return eStatus;
}
```

- Holding Registers 保持缓存器的 Callback Function，处理保持缓存器相关的指令。

```
eMBErrorCode eMBRegHoldingCB(UCHAR * pucRegBuffer, USHORT usAddress,
        USHORT usNRegs, eMBRegisterMode eMode)
{
    eMBErrorCode    eStatus = MB_ENOERR;
    USHORT          iRegIndex;
    USHORT *        pusRegHoldingBuf;
    USHORT          REG_HOLDING_START;
    USHORT          REG_HOLDING_NREGS;
    USHORT          usRegHoldStart;
```

```
    pusRegHoldingBuf = usSRegHoldBuf;
    REG_HOLDING_START = S_REG_HOLDING_START;
    REG_HOLDING_NREGS = S_REG_HOLDING_NREGS;
    usRegHoldStart = usSRegHoldStart;

    /* it already plus one in modbus function method. */
    usAddress--;

    if ((usAddress >= REG_HOLDING_START)
            && (usAddress + usNRegs <= REG_HOLDING_START + REG_HOLDING_NREGS))
    {
        iRegIndex = usAddress - usRegHoldStart;
        switch (eMode)
        {
        /* read current register values from the protocol stack. */
        case MB_REG_READ:
            while (usNRegs > 0)
            {
                *pucRegBuffer++ = (UCHAR) (pusRegHoldingBuf[iRegIndex] >> 8);
                *pucRegBuffer++ = (UCHAR) (pusRegHoldingBuf[iRegIndex] & 0xFF);
                iRegIndex++;
                usNRegs--;
            }
            break;

        /* write current register values with new values from the protocol stack. */
        case MB_REG_WRITE:
            while (usNRegs > 0)
            {
                pusRegHoldingBuf[iRegIndex] = *pucRegBuffer++ << 8;
                pusRegHoldingBuf[iRegIndex] |= *pucRegBuffer++;
                iRegIndex++;
                usNRegs--;
            }
            break;
        }
    }
    else
    {
        eStatus = MB_ENOREG;
    }
    return eStatus;
}
```

- Input Register 输入缓存器的 Callback Function，处理输入缓存器相关的指令。

```
eMBErrorCode eMBRegInputCB(UCHAR * pucRegBuffer, USHORT usAddress, USHORT usNRegs )
{
    eMBErrorCode    eStatus = MB_ENOERR;
    USHORT          iRegIndex;
    USHORT *        pusRegInputBuf;
    USHORT          REG_INPUT_START;
    USHORT          REG_INPUT_NREGS;
    USHORT          usRegInStart;
```

```
    pusRegInputBuf = usSRegInBuf;
    REG_INPUT_START = S_REG_INPUT_START;
    REG_INPUT_NREGS = S_REG_INPUT_NREGS;
    usRegInStart = usSRegInStart;

    /* it already plus one in modbus function method. */
    usAddress--;

    if ((usAddress >= REG_INPUT_START)
            && (usAddress + usNRegs <= REG_INPUT_START + REG_INPUT_NREGS))
    {
        iRegIndex = usAddress - usRegInStart;
        while (usNRegs > 0)
        {
            *pucRegBuffer++ = (UCHAR) (pusRegInputBuf[iRegIndex] >> 8);
            *pucRegBuffer++ = (UCHAR) (pusRegInputBuf[iRegIndex] & 0xFF);
            iRegIndex++;
            usNRegs--;
        }
    }
    else
    {
        eStatus = MB_ENOREG;
    }

    return eStatus;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M460 Series BSP CMSIS V3.00.001
- IDE 版本
  - Keil uVersion 5.37

3.2 硬件需求

- 电路组件
  - NuMaker-M467HJ V1.0
- 示意图
  - 将 UART0 TX (PB.13) pin 脚连接到 PC UART RX，并且以网络线连接 M467 与 PC 的网络孔。

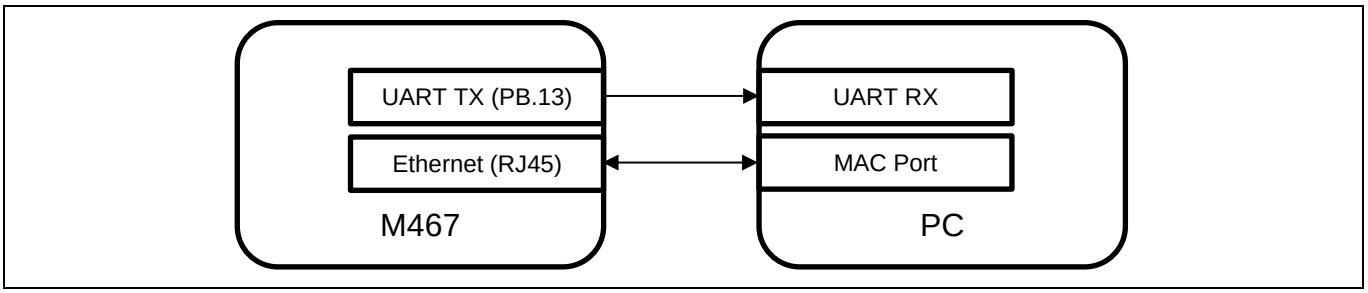

图 3-1 脚位连接示意图

4. 目录信息

EC_M467_LwIP_Modbus_TCP_V1.00	
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	Source file of example code
ThirdParty	
FreeRTOS	A real time operating system available for free download. Its official website is: http://www.freertos.org/
lwIP	A widely used open source TCP/IP stack designed for embedded systems. Its official website is: http://savannah.nongnu.org/projects/lwip/
modbus	FreeModbus Libary: A portable Modbus implementation for Modbus ASCII/RTU: https://www.embedded-experts.at/en/freemodbus/about/

图 4-1 目录信息

5. 范例代码执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *LwIP_Modbus_TCP.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
2023.06.24	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.