使用LwIP實現Modbus TCP功能

NuMicro® 32位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例程式使用 LwIP 程式庫來實作 Modbus TCP 的工業控制網路應用，使用者可以藉由 Modbus TCP 協議，透過網路伺服器控制 M467 的狀態。
BSP 版本	M460 Series BSP CMSIS V3.00.001
開發平台	NuMaker-M467HJ V1.0

1. 概述

本範例程式使用 LwIP 程式庫來實作 Modbus TCP 的工業控制網路應用，使用者可以藉由 Modbus TCP 協議，透過網路伺服器控制 M467 的狀態。

Modbus 是一種工業通訊協定，用於自動化系統中，具有監控系統和遠程數據收集等功能，Modbus 通訊協定常用的通訊介面包括 RS-232、RS-485 和乙太網等，它定義了一組命令和數據格式，使主機能夠向從機發送指令，這些指令可以用於讀取和寫入從機數據。

Modbus 主要用於監控和控制各種設備，如 PLC（可編程邏輯控制器）、傳感器、閥門、伺服驅動器和人機介面等。它提供了一種標準化的通訊方式，使不同供應商的設備能夠互相通訊，提高自動化系統的效率和可靠性。

1.1 原理

Modbus 協定定義了多種指令碼，用於控制從機的行為，以下是 Modbus 協定中常見的指令碼：

指令碼	指令名稱
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 指令碼

以下針對這些控制指令，說明主機如何控制從機：

- **Read Coils（讀取線圈）**：這個指令用於讀取一筆位元數據（通常表示狀態）。
- **Read Discrete Inputs（讀取離散輸入）**：這個指令類似於 Read Coils，主要用於讀取離散輸入的狀態，通常表示開關或按鈕的狀態。
- **Read Holding Registers（讀取保持暫存器）**：這個指令用於讀取 16 位元的數值數據。

- **Read Input Register（讀取輸入暫存器）**：這個指令類似於 Read Holding Registers，但是用於讀取輸入寄存器的值。輸入寄存器通常用於儲存感測器的數據。
- **Write Single Coil（寫入單個線圈）**：這個指令用於寫入單個位元數據。
- **Write Single Register（寫入單個暫存器）**：這個指令用於寫入單個 16 位元的數值數據。
- **Write Multiple Coils（寫入多個線圈）**：這個指令用於寫入多個位元數據到不同線圈中。
- **Write Multiple Registers（寫入多個暫存器）**：這個指令用於寫入多個 16 位元的數值數據到不同暫存器中。

這些控制指令提供了主機與從機之間的基本控制功能，可以用於監控和控制自動化系統中的各種設備，主機可以根據需要使用這些指令進行讀取或寫入操作，從而實現對從機設備的控制和調節。

1.2 執行結果

本範例程式執行編譯和燒錄後，使用 Modbus TCP 的 PC 工具 Modbus Poll 程式進行通訊

Modbus Poll 操作如下：

1. 點選 **Connect** 圖標，設定 IP 地址為 192.168.1.2，並且與 PC 位於相同網域。

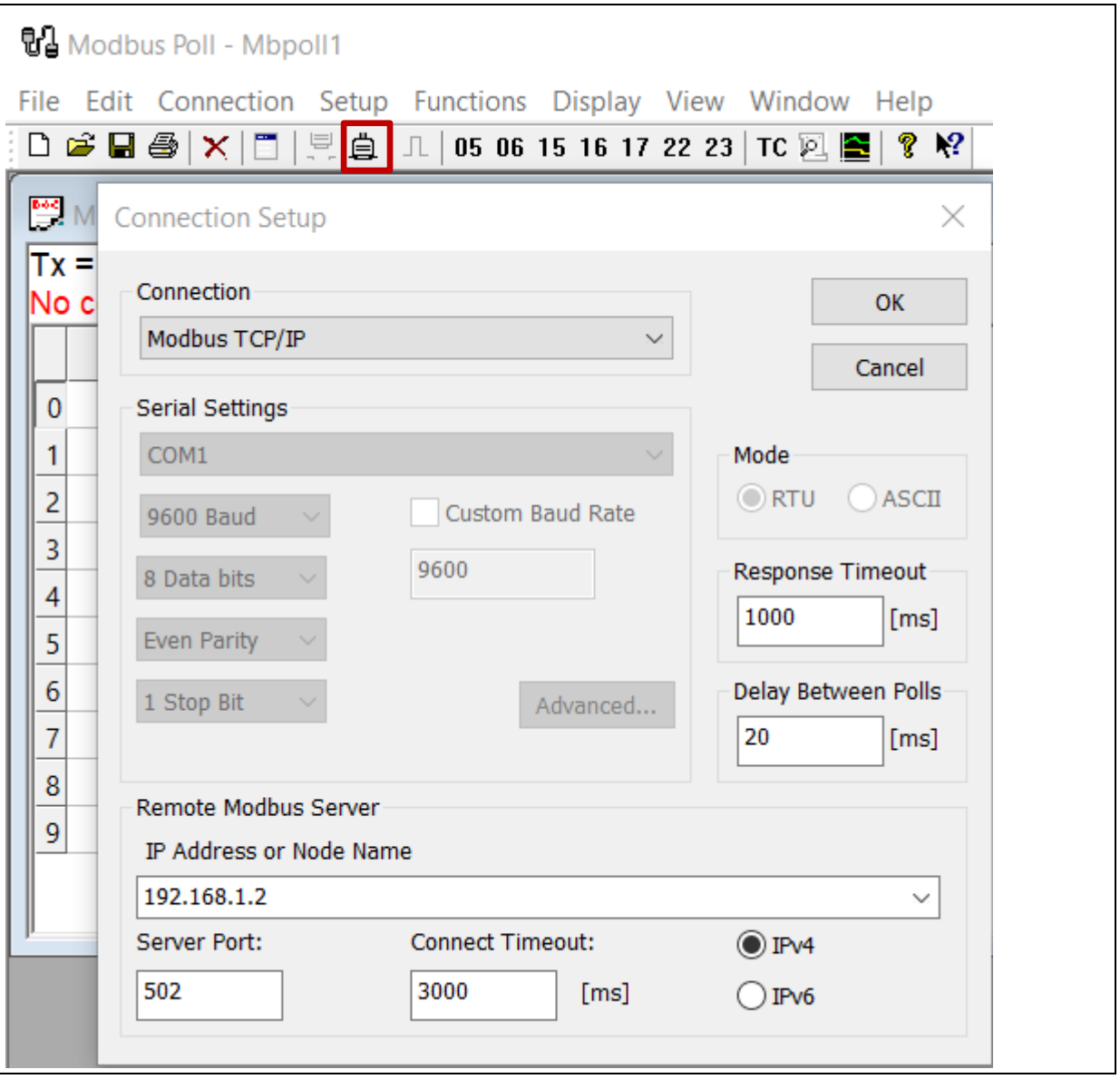

圖 1-1 連線設定

2. 點兩下欄位 0 的數值，可開啟指令視窗，設定 Value 為 151 並按下 **Send**，即可發送數據。

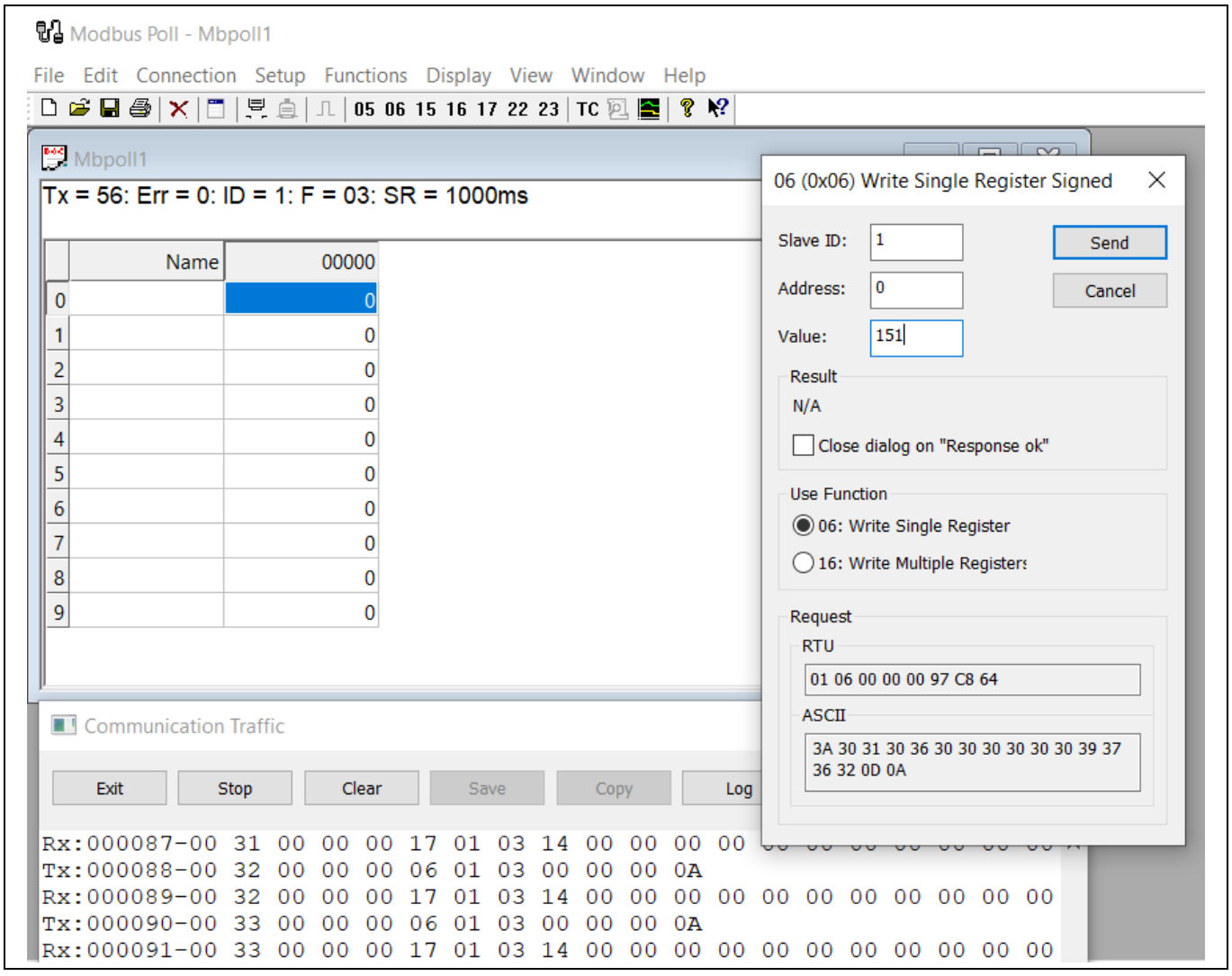

圖 1-2 發送數據

通訊結果如圖 1-3 所示。

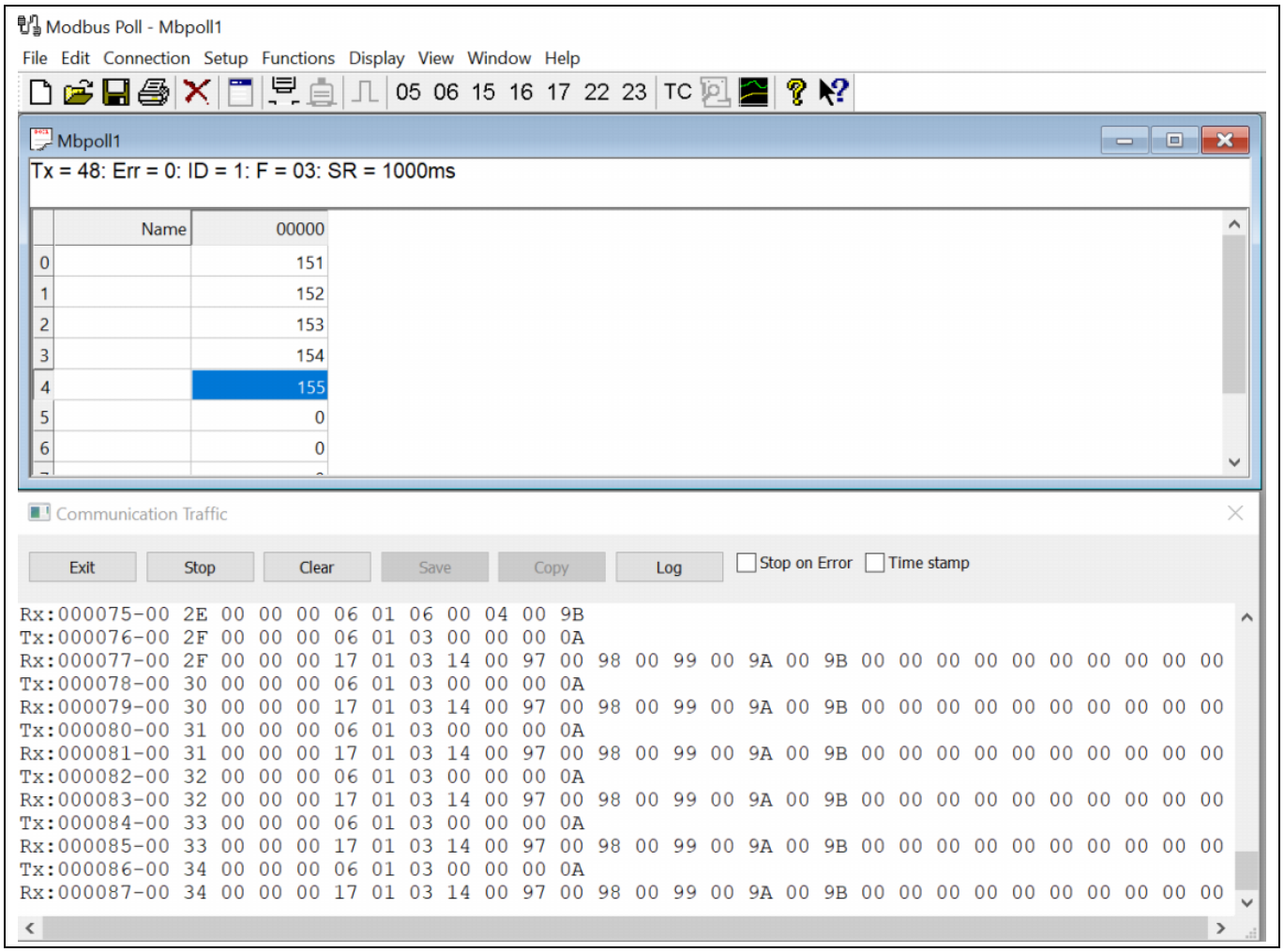

圖 1-3 Modbus TCP 通訊數據

2. 程式介紹

設定 MCU 硬體狀態，並建立 FreeRTOS 上的 TCP Task，執行網路應用，程式位於 *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( ;; );
}
```

初始化系統後，執行 TCP Task，設定網路 IP 等相關參數，並啟動 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(  );
    }
}
```

底下接著介紹 FreeModbus 程式庫提供的指令處理程式：

- Coil 線圈的 Callback Function，處理線圈相關的指令。

```
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 離散輸入的 Callback Function，處理離散輸入相關的指令。

```
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 保持暫存器的 Callback Function，處理保持暫存器相關的指令。

```
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 輸入暫存器的 Callback Function，處理輸入暫存器相關的指令。

```
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. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
  - NuMaker-M467HJ V1.0
- 示意圖
  - 將 UART0 TX (PB.13) pin 腳連接到 PC UART RX，並且以網路線連接 M467 與 PC 的網路孔。

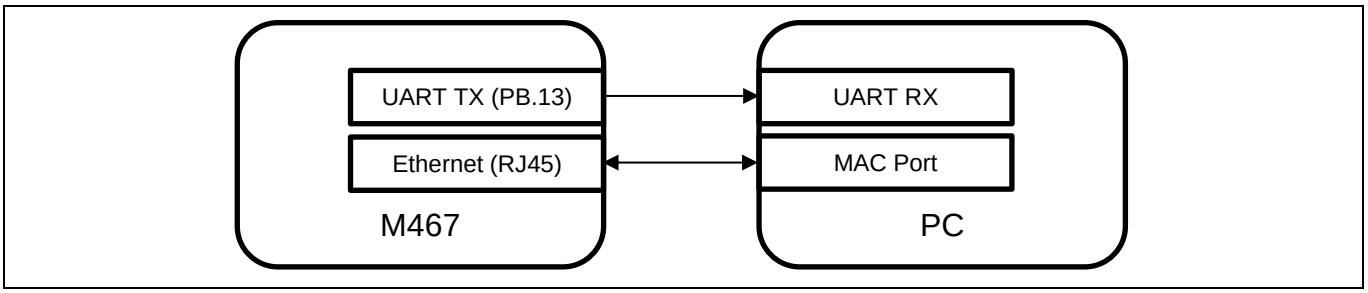

圖 3-1 腳位連接示意圖

4. 目錄資訊

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/

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *LwIP_Modbus_TCP.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2023.06.24	1.00	初始發布。

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.